jamk.fi

Scalable Multiplayer Components

Rami Ojala

Bachelor's thesis
April 2020
School of Technology
Degree Programme in Information and Communications Technology

jamk.fi

Description

Author(s) Ojala, Rami	Type of publication Bachelor's thesis	Date April 2020
		Language of publication: English
	Number of pages 63	Permission for web publication: x
Title of publication **Scalable Multiplayer Components**		
Degree programme Information and Communications Technology		
Supervisor(s) Ari Rantala; Tuula Kotikoski		
Assigned by		

Abstract

Objectives for this thesis was to create a scalable multiplayer environment and a prototype game with it. Additional tasks were to learn new technologies and improve understanding on already familiar technologies. Main goals were to learn new and apply old information, and possibly create a base for a future company.

The implementation method became naturally a functional development / research, as there was only a basic idea at the beginning; that was later researched and developed.

As a result, the author was able to create a user creation website and an authentication pipeline for a game client, a written protocol for game client and login server communications, and a networked 22-computer development environment that can be used to develop the system this thesis started.

This thesis was not able to reach the full environment and a game made with it, which disappointed the author. As a positive note a lot of new information about new technologies and thoughts on how to use those in the future development was learned. The possible company is not happening as of now, since the result of this thesis is not significant enough to merit a company.

Keywords/tags (subjects)
Scalable multiplayer environment, Multiplayer, Scalability, System, Authentication

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t) Ojala, Rami	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Huhtikuu 2020
	Sivumäärä 63	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x
Työn nimi **Skaalaautuvia moninpelikomponentteja**		
Tutkinto-ohjelma Tieto- ja viestintätekniikka		
Työn ohjaaja(t) Ari Rantala; Tuula Kotikoski		
Toimeksiantaja(t)		

Tiivistelmä

Tehtävänä opinnäytetyössä oli luoda skaalautuva moninpeliympäristö sekä prototyyppi peliympäristön päälle. Lisäksi tehtävänä oli tutustua uusiin teknologioihin ja syventää osaamista jo tuttuihin teknologioihin. Päätavoitteena oli oppia uutta ja soveltaa mahdollisimman paljon jo opittua asiaa sekä mahdollisesti luoda pohjaa tulevalle yritykselle.

Tutkimustavaksi muodostui luontaisesti kehittämistyö/-tutkimus, sillä työn alussa oli vain pelkkä idea, jota lähdettiin tutkimaan, jalostamaan ja lopuksi kehittämään sovellukseksi.

Lopputuloksena syntyi internetsivusto tilin luonnille, pelin sisäinen tunnistautumisjärjestelmä sekä 22-tietokoneen verkotettu kehitysympärsisö. Lisäksi aikaan saatiin kirjallinen protokolla, jonka avulla voitiin tunnistautua kahden eri ohjelmointikielen kesken.

Työssä ei päästy valmiiseen järjestelmään, joka oli kova pettymys työn kirjoittajalle. Positiivista kuitenkin on, että työtä tehtäessä opittiin todella paljon uusia teknologioita ja ajatuksia siitä, miten näitä voitaisiin jatkossa käyttää järjestelmän jatkokehitykseen. Omaa yritystä aiheesta ei ainakaan toistaiseksi ole tulossa, sillä saavutettu tulos ei riitä oman yrityksen perustamiseen.

Avainsanat (asiasanat)
Skaalautuva moninpeli järjestelmä, Moninpelit, Skaalautuvuus, Järjestelmä, Tunnistautuminen

Muut tiedot (salassa pidettävät liitteet)

Contents

Figures

Terminology

API	Application programming interface
ARM	Advanced RISC machines, microprocessor architecture
DDoS	Distributed denial of service
DoS	Denial of service
DIY	Do it yourself
JSON	JavaScript object notation
MVC	Model view controller
NODE	A single unit in a group of units, I.e. computer in a cluster
OOP	Object oriented programming
OS	Operating system
PSU	Power supply unit
RPC	Remote procedure call
SSL	Secure sockets layer
STUB	A limited fake implementation of a component that is expected to do something, and just does it by faking it.
TLS	Transport layer security
URL	Uniform resource locator

1 Introduction

1.1 Multiplayer games

The trendline on people playing video games is rising approximately by one hundred million active gamers a year, estimating 2.6 billion gamers in the year 2020 (Number of active video gamers worldwide from 2014 to 2021. 2017.). The number of games being published is also rising rapidly, and the year 2018 alone saw the release of 9,050 different publications in Steam alone from companies across the games industry (ibid.). While no real statistics can be found on how many games from all the games released in said year have an online component, quickly calculating from Steam, year 2018 saw 630 games with a multiplayer tag associated with it (Games with Multiplayer tag released in 2018 in Steam platform 2020), even adding two hundred to count for left out entries, which is still less than 10% from the year 2018 total of 9,050 publications.

Tools such as Unity and Unreal Engine have come a long way in these past 10 to 15 years, and the amount of games released is a proof of that. Many of these tools, however, focus on graphical fidelity and ease of use for the developer, which is good; however, this leaves the online components and learning resources for the software to be desired.

1.2 Goals

This thesis is an attempt to create an online system or parts of it that can be attached next to engines such as Unity or Unreal that allow a connection to an online server with a set of protocols allowing a game client to communicate with said server. Ideally the thesis has a complete multiplayer system with a Unity prototype game on top of it.

The goal of the thesis itself, however, is not the system itself but to learn and apply as much knowledge of multiplayer systems as possible in order to hopefully in the future build the system started here completely.

To achieve these goals the author researched manuals, asked many questions from teachers, engineers and experienced game industry employees, while trying to simultaneously implement things based on the gathered information. The research method of this thesis is a mix of development, research and studying failures, relying on existing information from various sources and observed behavior of the system under failure.

2 Used and Researched Technologies

2.1 C++

C++ is one of the first object-oriented programming (OOP) languages that saw its first release somewhere between 1979 and 1983 while being called "C with Classes". It was renamed "C++" in 1983 after the incremental "++" operator. (History of C++. n.d.)

Year 1985 saw the first release of a C++ reference book, called "The C++ Programming language" written by C++ creator Bjarne Stroustrup. (History of C++. n.d.)

The first C++ compiler was called "Cfront" and it compiled C++ code to plain C language. It was in use up to 1993 before it was abandoned due to difficulties implementing C++ exceptions. (History of C++. n.d.)

The first standardized version of C++ was released in 1998 by the International Standardization Organization and was called "C++ ISO/IEC 14882:1998" or "C++98". The ISO organization has then later updated the standard in 2003, 2011, 2014 and the latest update is from 2017. (ISO/IEC 14882:1998. 1998.)

C++ at some point was to be the main programming language of this thesis because of its low-level implementation and performance compared to languages like Python or C#. Originally, this thesis was supposed to be completely on top of Go programming language, but the author decided to use C++ at this point, due to maximum performance attitude.

The first chat server prototype was created using a C++ language and Epoll C++ Linux library; however, the author quickly noticed that securing the sockets with OpenSSL or LibreSSL was too much of a hassle and Go was selected as a main programming language instead.

2.2 Epoll

Epoll is a Unix C++ socket extension library that allows the computer to monitor multiple File Descriptors (or sockets) simultaneously on a single CPU thread. It is like Poll or Select libraries; however, it is much faster when handling a large number of sockets. (Linux Programmer's manual 2019.)

Epoll was the solution to a question “does it scale up to thousands” by the author, since Poll and Select seemed to have issues after reaching a certain number of packages. (Results for dphttpd SMP n.d.)

2.3 OpenSSL

OpenSSL is an Apache 2.0 licensed encryption library implementing SSL and TLS protocols for C++. It is used to encrypt network communications between software and was first released in 1998 from the remains of SSLeay library. It has seen constant updates since and currently it is overseen by a management committee and developed actively by a team of committers. OpenSSL is funded by donations from sponsors and sometimes companies donate man hours. (Cox 2018.)

OpenSSL had a notorious reputation when researching the chat server security, with a lot of bugs and complicated code due to outdated compatibility features and programming standards was the most common opinion when trying to decide how to secure the chat server.

2.4 LibreSSL

LibreSSL is developed by The OpenBSD project, an open-source project that is producing OS called OpenBSD and has adopted multiple other projects since including LibreSSL (OpenBSD 2019.). LibreSSL is a fork from OpenSSL from year 2014, with the goal to modernize the code, improve security and apply best practices to development. (LibreSSL 2019.)

LibreSSL has removed several OpenSSL features and deprecated code, which results in less bugs, and most OpenSSL bugs discovered are, due to this, rarely if ever found in LibreSSL. (LibreSSL 2016.)

LibreSSL was the choice to secure chat server network traffic, but the lack of documentation, good sources and the authors personal lack of knowledge in C++, the progress to implement LibreSSL into the chat server was slow. Step in Google's Go language.

2.5 Google's Go language

Go is an open source procedural programming language developed by Robert Griesemer, Rob Pike, and Ken Thompson from Google (Yadav n.d.), which means that it ditches every object oriented aspect that C++ has adopted to C and loosely bases the syntax on C (Golang 2017.).

Go started development in 2007 and was first published in 2009. The core focuses for the development were on simplicity of dynamic languages (dynamical language is like

Python or JavaScript) and the efficiency and safety of statically typed languages (like C). Go was also designed from ground up for multicore CPUs in mind. (Yadav n.d.)

After constantly failing with C++ and LibreSSL, the author decided to try out Go programming language after getting a recommendation from an engineer working at JAMK university of Applied Sciences. The switch to Go helped, and the progress was a lot faster; due to Go's code being simpler but also because Go has a lot of the security features already built into it. This meant that the author did not have to fight with OpenSSL or LibreSSL which sped the development significantly. Go also introduced channels, wait groups and goroutines that made coding concurrent code a breeze.

2.6 Docker

Docker is an application that wraps programs in Dockerfiles and runs them as "Containers". These containers have a minimal Linux installed on them and the requirements for the application to run. These containers then can be setup and deployed rapidly across multiple devices manually or using a software like Ansible. (Docker overview n.d.)

Each Dockerfile holds commands to setup a single service that can be duplicated inside the Docker engine multiple times. Dockerfile requires a base Linux image to build on top of it; in this case it was Alpine Linux. This Dockerfile is then built to a Docker image, which can in turn be stored in Dockerhub and replicated to as many computers as necessary. (Docker overview n.d.)

Go applications were easy to build and deploy using Docker; this meant that a single service like login server could be deployed ten times, and load balanced using HAProxy in a fraction of the time it would take otherwise. This in turn meant that the said service became infinitely more scalable than before. This is called sideways scalability. Docker also has a good synergy with applications like Vagrant or Ansible.

2.7 HAProxy

HAProxy is a single threaded software that can be used as proxy, reverse-proxy, SSL-terminator, and this case protection against DDoS and a load balancer. (What HAProxy is and isn't n.d.)

There are many use cases for HAProxy inside an MMO environment, one being a load balancer between website instances, and another being load balancer between web-site instances and their database nodes.

Some notable use cases could be an SSL terminator, DDoS protection and a Tarpit for spammy connection attempts but these have not been researched by the author.

2.8 YAML

YAML is a human readable Data Serialization standard, where scope of data is in-dented using spaces. Most notable YAML user is probably Minecraft that uses YAML to serialize its configuration files. (Ben-Kiki, Evans & döt Net 2009.)

YAML is used in some software as configuration serializing. The author researched for YAML briefly for Go languages configuration serializing, but chose JSON instead for Go languages serializing library.

2.9 Ansible

Ansible is a software developed with Python that allows running of YAML notated scripts called “Playbooks” that deploy software over SSH connections, i.e. a single lo-cal laptop can be used to deploy and set up several machines inside Google Cloud service. Combined with Docker, Ansible can be used to deploy complete networked services over SSH in local cluster or clouds. (How Ansible Works n.d.)

The author had limited overlapping time to research Ansible over a course called “Automation of Services” taught by Jarmo Viinikanoja at JAMK University of Applied Sciences. This led the author to believe that Ansible could be used against a Cloud or a Networked cluster of computers to almost instantly set up an MMO service defined in a single playbook.

Playbooks could be created from small parts of the Networked system or the whole system could be defined in a single Playbook, this would make the system admins work a lot easier.

2.10 Vagrant

Vagrant is a software used to control virtual machines; almost like Ansible but for virtual machines. Vagrant also uses YAML notation in its Vagrantfile to execute the commands specified by the user. The main reason for Vagrant’s existence is to make identical development environments for software developers across the world. This way, the developed software runs no matter what is used to develop it with. (Introduction to Vagrant n.d.)

Vagrant was researched and tested during Automation of Services course during studies at JAMK. The author created a single Vagrantfile that would set up the authentication pipeline with a single command as virtual machines.

2.11 Linux Ubuntu

Ubuntu is a Debian Linux based operating system first released in October 2004, and it is developed by a group called Canonical started by Mark Shuttleworth. Ubuntu gets a new release every six months, and every fourth release gets a long-term support usually marked with “LTS”. (The story of Ubuntu n.d.)

Ubuntu has many promises on its “Ubuntu Manifesto”, including that it is always free; also that it is shipped with stable and regular release cycles and it is entirely committed to the principles of open source. (Welcome to ubuntu n.d.)

Ubuntu worked as a base operating system for all the virtual and non-virtual machines for this thesis with Docker containers being the only exceptions using Alpine Linux. Ubuntu was chosen for the LTS supported releases, and the amount of documentation and user support available on websites such as Stack Overflow.

2.12 Linux Alpine

Alpine Linux in the scope of this thesis means the “Mini Root Filesystem” version of Alpine Linux used as a base for Docker containers.

Alpine Linux is a minimal Linux designed with security, size and simplicity in mind and it is built on top of musl libc and busybox (Small. Simple. Secure n.d.). The most significant difference compared to Ubuntu is that it uses APK package manager instead of APT package manager that Ubuntu uses.

Alpine worked as a base for every Dockerfile the author created, because of its light weight and simplicity.

2.13 Oracle VM VirtualBox

Oracle VM Virtualbox is an operating system virtualization tool that builds another OS on top of the Host operating system Hypervisor, adding another operating system between the executable software and the host OS, thus allowing Linux apps to run on windows and vice versa. (Docker vs Virtual Machine – Understanding the Differences 2019.)

VirtualBox was used, when testing parts of the system locally in a single computer. Even a Vagrant file was created as a part of the Automation of Services course, to quickly set up a virtual machine environment for the different components in authentication pipeline.

2.14 Visual Studio Code

Visual Studio Code is an opensource code editor built with JavaScript on top of Electron.js. It can be used to program almost anything and extended with plugins to do exactly what the developer wants. (Getting Started n.d.)

Visual Studio Code text editor was chosen as coding platform simply because of its versatility and portability. Some development happens on Windows and Some on Linux; and this made Visual Studio Code excellent choice, as it kept the environment mostly the same when developing in Windows or Ubuntu.

2.15 Google's Protocol Buffers

Google's protocol buffers are scripts that look similar to C/C++ structs. These scripts are built using protocol buffer compiler that can build ".proto" files to C++, C#, Go and Python (and others) code files. These code files can be used inside a code base to serialize, and deserialize messages to and from binary for network traffic. (Developer Guide 2019.)

Protocol buffers are used in many games to serialize protocol messages between server and client, one of such games is Pokemon Go. In Pokemon Go's case these messages were intercepted using a proxy server (or man in the middle) and deserialized quickly, and an illegal Pokemon Go real time map was created using this information. (Pankoff 2016.)

Google's Protocol Buffers was the first attempt to implement a client server protocol for login server. The shear simplicity and versatility of this system would have saved a lot of time when writing the login server but as Murphy's law would have it, the compatibility of Protocol Buffers and Mono C# was broken, and after some time trying to get it to work with Unity the author gave up and wrote their own protocol.

2.16 C# / Mono

C# is the 4th most used programming language 2020, right behind JavaScript (Roper 2020.). It was developed by a team in Microsoft, and it was release in 2002. C# can be used to create web applications, desktop applications, games and console applications, you name it. (Srivastava 2017.)

Mono is open source .NET framework-compatible programming language that takes the C# standard and implements a cross platform version like C#. This enables the use of C# like syntax on cross platform applications but in a sense, it is not the same as C#. (Adams 2001.)

Unity uses Mono, so when a software library says it is compatible with C#, it doesn't automatically mean that it is compatible with Mono, as the Author noticed trying to integrate Google's Protocol Buffers while creating first version of the Login pipeline protocols.

2.17 Unity

Unity is 2D capable 3D game engine used by 62 developers out of 100 inside United Kingdom in 2014 (What game engines do you currently use? 2014.). Unity or Unity3d as it was called at first, was a brainchild of Nicholas Francis, Joachim Ante and as a late entry to the team David Helgason. Official Unity release was in 2005 and it has gained huge success since. (Haas n.d.)

Popularity of Unity can be tracked back to few influencers called Tornado Twins that had a popular tutorial series of Unity3d trough 2009 and 2014 (Make a Videogame from Scratch - Tutorials 2014.), and the fact that some bigger corporations, like CCP

Games had used Unity3d internally for fast prototyping of features (Team Avatar and the future of our prototype. 2012.).

Unity can be used in MMO environment, to create the whole client for any MMO. In this case, Unity was used to create a login client prototype, for quickly testing the login pipeline.

2.18 Google’s gRPC

gRPC is Google’s implementation of an RPC architecture that allows a client to connect to an RPC server, and call functions stored inside that server. This allows critical functions to run tamper free inside an RPC server, and return the data to the client. (Chalin, Perkins, & Agung 2020.)

RPC architecture in MMO environment could be used, to implement critical functions inside the system, so that they cannot be reverse engineered from the client software; also RPC functions could be used so that the server sends RPC calls to game client, this would make the game client an RPC server and connection handler or world server an RPC client.

Some early ideas for the authentication pipeline were considered on top of gRPC and RPC, but were dismissed over a more hands on approach; and in favor of a proper protocol.

2.19 Git

Git is an open source; source code version control created by Linus Torvalds in 2005. Git is popular version control, and it is used by most open source projects in Git services such as GitLab or GitHub. With Git, a project can be rolled back to any point in time if a flaw has been introduced or a mistake made; with Git you can also separate your development to branches and have multiple people work on the project at the same time. (What is Git n.d.)

Some popular open source projects on Git are Visual Studio Code with approximately 19,100 unique contributors, and Flutter which has approximately 13,000 unique contributors. (Most popular GitHub open source projects worldwide in 2019, by number of contributors (in 1,000s)* 2019.)

Git can be more than just a source code version control if hosted locally. It can be used to version control pretty much anything, including game art assets, audio assets and story scripts.

2.20 Django

Django is a web development framework created using Python. Django uses a common MVC architecture to map URL paths to different views; it comes out of the box with SQLite database, but it can be changed to use databases such as MariaDB or MySQL. (Django at a glance n.d.)

Any Multiplayer environment needs a way to create a user account. In the past in games like Diablo 1 and Diablo 2 the account was created inside the game client, but these days a website is more common than an in-game form. A Django website can be used to manage user data, promote in game events, host a dev blog or even host a store; possibilities are limitless.

2.21 Python

Python is dynamic interpreted object-oriented programming language released by Guido van Rossum in 1991 (Python Introduction n.d.). Python is like Java, Perl or Ruby; and focuses on simplicity, portability and ease of use. (Beginners Guide 2019.)

Thanks to Python’s ease of use and simplicity, it is the most used programming language today. Holding top 1 position over languages like Java, JavaScript, C# or even PHP and C/C++ with whopping near 30% share. (Roper 2020.)

Python was used as a part of Django for the user registration website. Other use cases are limited for Python, because of its dynamic nature and relatively slow performance compared to Go or C++.

2.22 MySQL

MySQL is the most used database in amongst developers (Most popular database technologies among developers worldwide as of January 2019. 2019.); it is a relational database like MariaDB and is the version developed by Oracle, after the acquisition from Sun Microsystems in 2009 (10 reasons to migrate to MariaDB (if still using MySQL) 2015.).

MySQL was first chosen as the database for this thesis, after “Advanced Databases” course taught by Jouni Huotari at JAMK University of Applied Sciences. During the course, a topic of Data replication came up which lead the author down a rabbit hole that eventually lead to MariaDB. MySQL Data Replication Required a several different installations of Linux packages, while MariaDB had these built in.

2.23 MariaDB

MariaDB is the 7th most used database among developers (Most popular database technologies among developers worldwide as of January 2019. 2019.). MariaDB is a Fork from MySQL database that was released in 1995. MySQL AB company was sold to Sun Microsystems in 2008 which in turn was sold to Oracle in 2009. This prompted the original creators to fork MySQL in 2009 and call it MariaDB out of distrust of Sun Microsystems that at the time was the biggest competitor to MySQL with its Oracle Database. (10 reasons to migrate to MariaDB (if still using MySQL) 2015.)

MariaDB has been designed to be a drop-in replacement for MySQL (MariaDB versus MySQL: Compatibility n.d.) and boasts many improvements over its predecessor; features like more storage engines, speed improvements and multi-source replication. (MariaDB versus MySQL - Features n.d.)

MariaDB was researched together with Galera Cluster and is the database for Frontend website. It replaced MySQL very early on due to Galera Cluster integrations and it also had a performance boost as a bonus which suited for the performance greed of the thesis.

2.24 Galera Cluster

Galera is a database replication tool for MySQL and MariaDB; it is used to share load between database nodes, and replicate data over nodes in the cluster as a data redundancy tool. With each node, the galera cluster can serve requests and take in additional data from user and share it in the cluster, where all the nodes can then have that information. (Galera introduction n.d.)

MariaDB Galera Cluster is highly scalable sideways, as each of its nodes can be used as an access point. Each node then could be load balanced using HAProxy and made highly available using Keepalived. Even the size of the database would not be a problem if using a network filesystem like GlusterFS. Galera Cluster was researched and planned for both the website and the games database, but it was not implemented in time due to Docker and Galera Cluster difficulties.

2.25 GlusterFS

GlusterFS is a free open source network filesystem, used in clouds and networked solutions. GlusterFS filesystem can be accessed as a storage drive across the network, and can be secured using TLS certifications so only users with access can connect to it. (GlusterFS Documentation n.d.)

GlusterFS storage can be a single hard drive on the server which is shared over the network or it can made highly redundant, using different raid setups or external cards to make sure no data is lost in case of emergency. What makes GlusterFS even more powerful is that it can be replicated over server instances, which means you can have any number of server computers across the cloud, and the data is automatically shared between those servers. (Architecture n.d.)

Certification encrypted traffic, and the highly redundant nature of GlusterFS, makes it a perfect tool for network storage in clouds and networked solutions like MMO environments. One use case would be to have two instances of GlusterFS replicate to one another, and use MariaDB with Galera Cluster to dump node data to that GlusterFS file system; in this case if the Galera Cluster ever fails to a point that MariaDB servers themselves are unsalvageable, the data is still secure and can still be reconstructed as a new MariaDB Galera Cluster using the latest node data stored in GlusterFS.

2.26 Keepalived

Keepalived is a C based Linux program that provides a failover to a second similar instance in case the first one fails (What is Keepalived ? n.d.). In example Keepalived is configured to monitor Apache service in a Linux computer, and if Apache service is not found on the master machine, then Keepalived will see that, and failovers to a secondary machine. Keepalived will keep monitoring the Apache service in master server, and will return the control to it if it sees Apache back in working condition. (Ellingwood 2015.)

In MMO environment, Keepalived could be used to monitor HAProxy load balancers, and failover to a slave instance in cases where the master falls. This helps at keeping the system available, in cases where HAProxy might fail. This was researched as a part of the website implementation, to make sure that load balancing was always available and websites reachable.

2.27 mermaid

mermaid is a JavaScript library that allows generation of diagrams and flow-charts from a markdown like syntax. This helps when designing or documenting a software flow or a system flow chart. (Lasn 2019.)

mermaid was used during the design phase of the login server component (Appendix 1), to help visualize the path a single connection attempt takes, and what are the values given and returned in those steps. Furthermore, mermaid can be used to visualize communication between a client and server to help develop a proper protocol.

2.28 REST

REST or "Representational State Transfer" is a software development architecture solution that helps a networked service to communicate certain states between its different components that might be on different servers. REST API is usually a HTTP/S server that is a single known constant in the system and can be used as a message delivery system between components. REST API can use POST HTTP headers (or GET) to return requested data or store requested data. (Avraham 2017.)

In MMO environments, REST API can be used a lot of different ways. In this case, REST API was used between a login server and a connection handler stub. Connection handler would update REST server with how many connections (and other relevant data like IP and Port that could be used trough REST API) it had, and then login server would receive a client connection, and pass that client the connection information of the least connected connection handler from REST API. Login server would also update temporary data to REST API, like login token and username that the connection handler would be able to confirm, when a new client connects to a connection handler.

REST API could be used as a system monitoring tool, where each part of the system sends updates with relevant information to the REST API, and REST API would have a route that parses the REST data to a meaningful view of the status of the system.

3 Project

3.1 Designs

Early designs for the required system below the service went through many changes during the design and implementation phase of the project. Early on, the author did not account for a website e.g. as seen in Figure 1, which was added later in the progress.

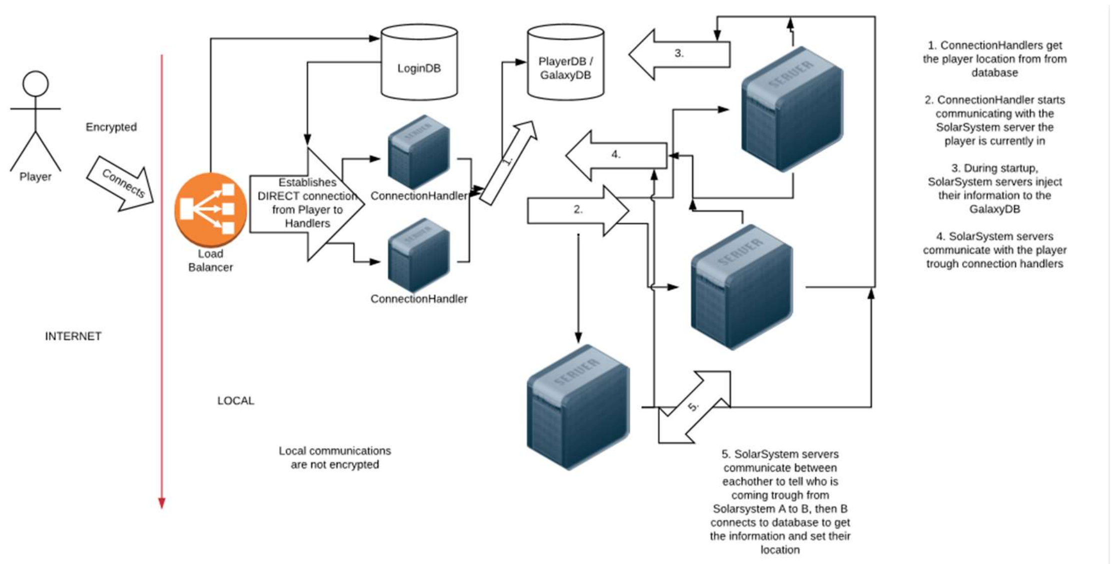

Player
Encrypted
Connects
INTERNET
Load Balancer
LOCAL
LoginDB
Establishes DIRECT connection from Player to Handlers
ConnectionHandler
ConnectionHandler
Local communications are not encrypted
1.
PlayerDB / GalaxyDB
2.
3.
4.
5.
SERVER
5. SolarSystem servers communicate between eachother to tell who is coming trough from Solarsystem A to B, then B connects to database to get the information and set their location
1. ConnectionHandlers get the player location from from database
2. ConnectionHandler starts communicating with the SolarSystem server the player is currently in
3. During startup, SolarSystem servers inject their information to the GalaxyDB
4. SolarSystem servers communicate with the player trough connection handlers

Figure 1. Early UML designs for the underlying system

After some research in the beginning, it was apparent that the hardware required for the system under the service would consist of more computers than the author had access to. This problem could be avoided for the start of the project, since early prototypes and components could be developed in a single computer with virtual machines. At an early stage, the author estimated a need for ten computers for the system illustrated in Figure 2. Closing the project, the computer count was even higher because a chat system was planned and databases were changed to clusters as illustrated in Figure 3.

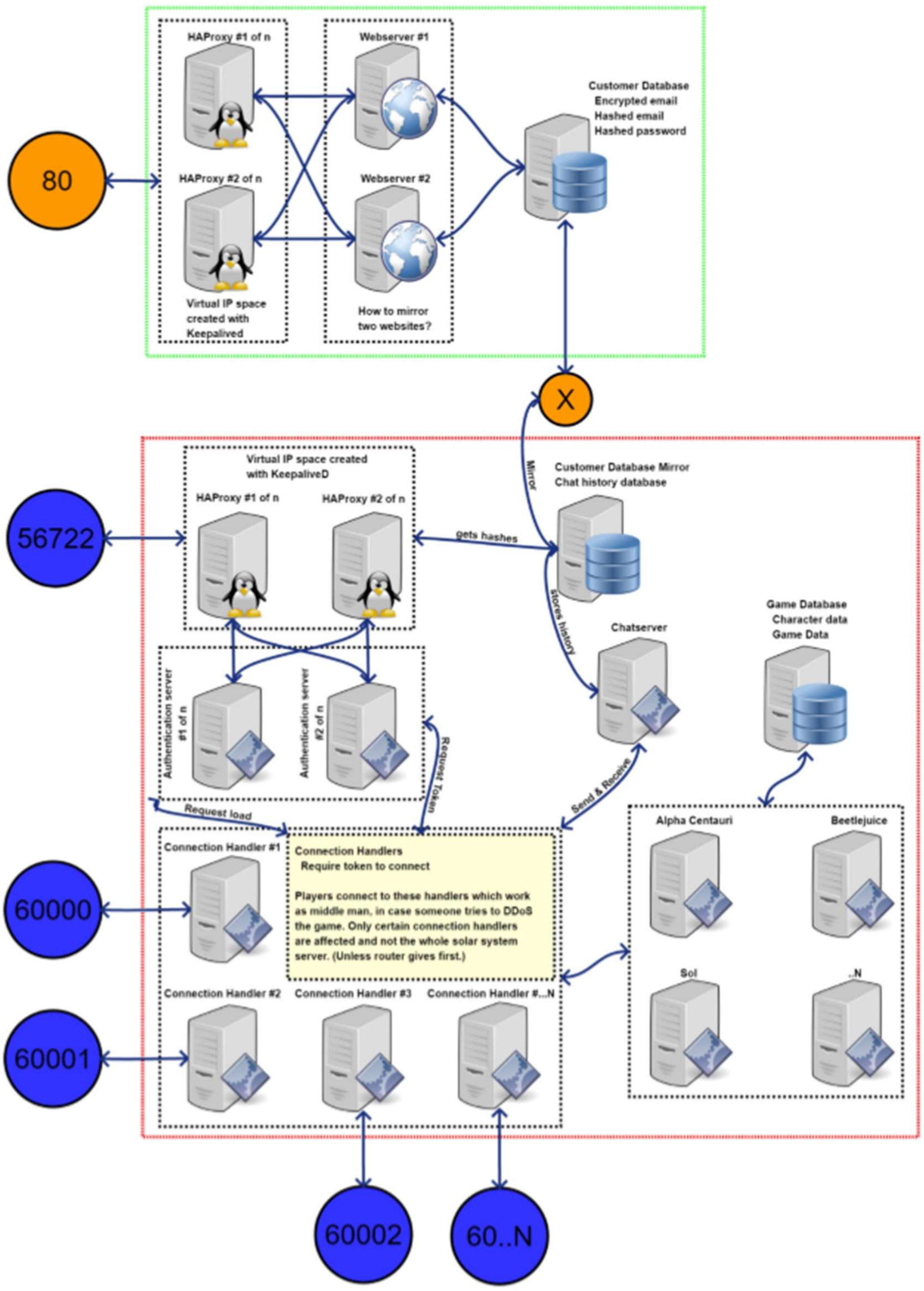


Figure 2. Second system estimation

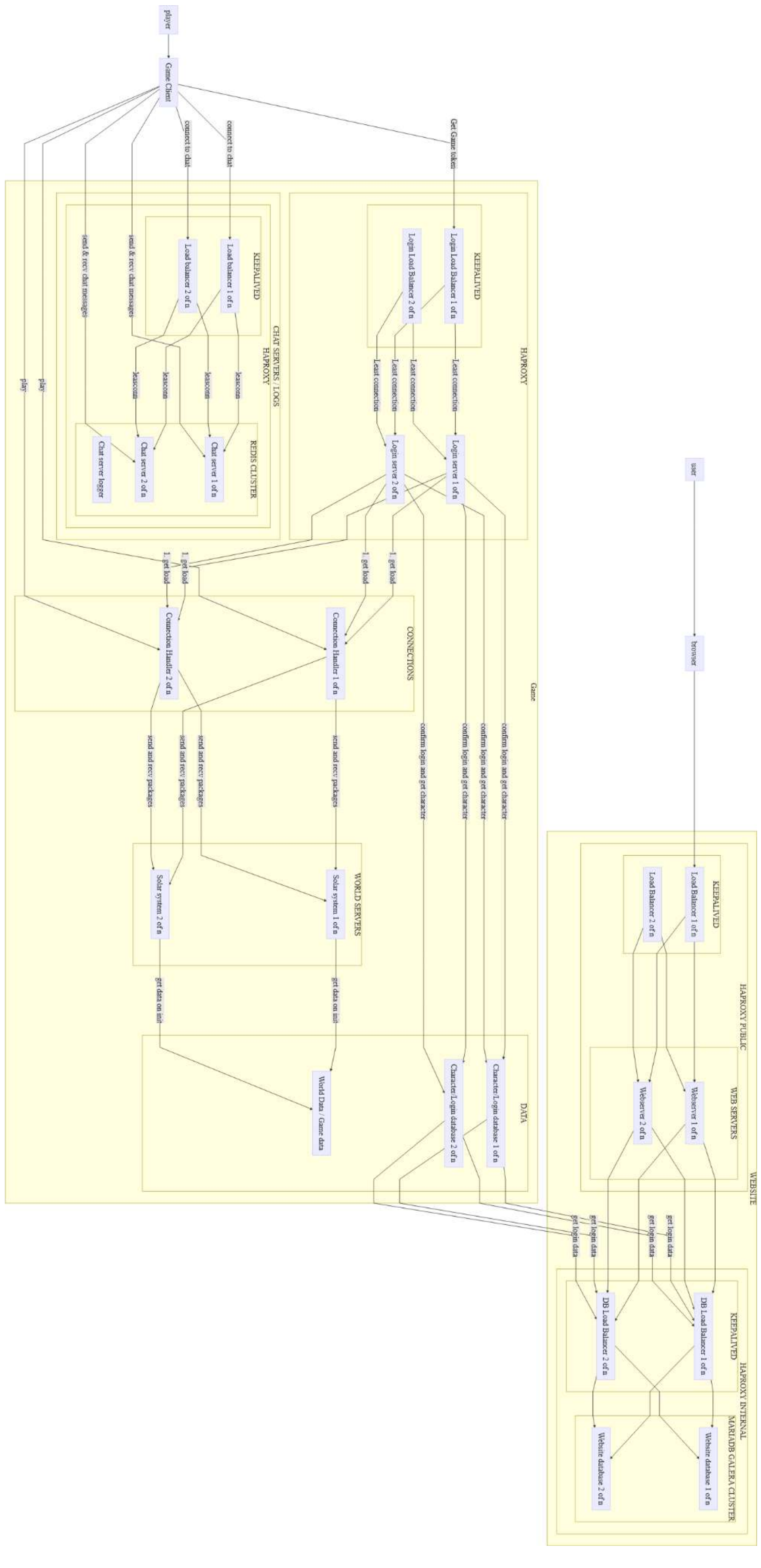


Figure 3. System diagram at the end of the project

3.2 Implementations

3.2.1 Server hardware and rack

Hardware was one of the bigger problems the author had to handle, as the planned system was large. Idea at the beginning was to use Raspberry Pi Zero single board computers to simulate the system as they are cheap and easily available, but this idea was later scrapped due to some free hardware.

Hardware problems were solved over the period of two summers, during which JAMK University of Applied Sciences was throwing out old computers and gave them away freely for anyone willing to take them. Computers in Figure 4, were salvaged as a cost cutting measure and to get other than ARM based processors for the system, since the ARM processor architecture is different from Intel and AMD 64 bit architectures and hence, compatibility was a concern. Over the two summers, the author was able to salvage 22 computers and a 24-slot network switch, which allowed the creation of a custom DIY server rack for the 22-computer network.

Figure 4. Free computers at JAMK

After solving the hardware crisis, the author faced another problem. The total of 22 computers took plenty of space, which the author did not have. This issue was solved by making a DIY server rack shown in Figure 5 from a garage shelf. The shelf was mounted firmly to a base with wheels for easy moving as shown in Figure 6. With this shelf, the whole system fitted to a small student apartment.

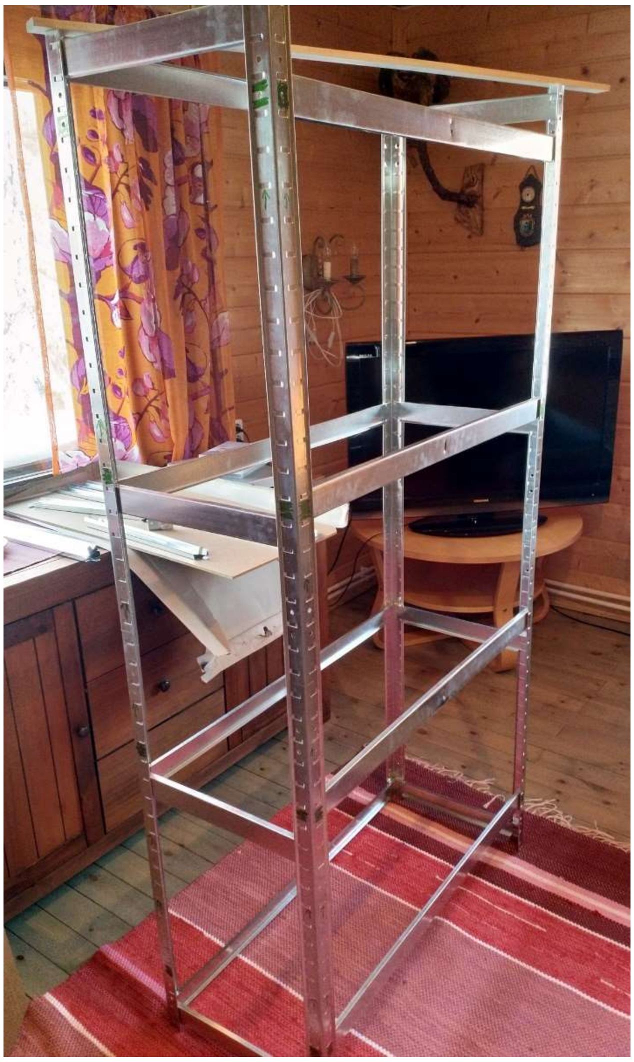


Figure 5. DIY server rack

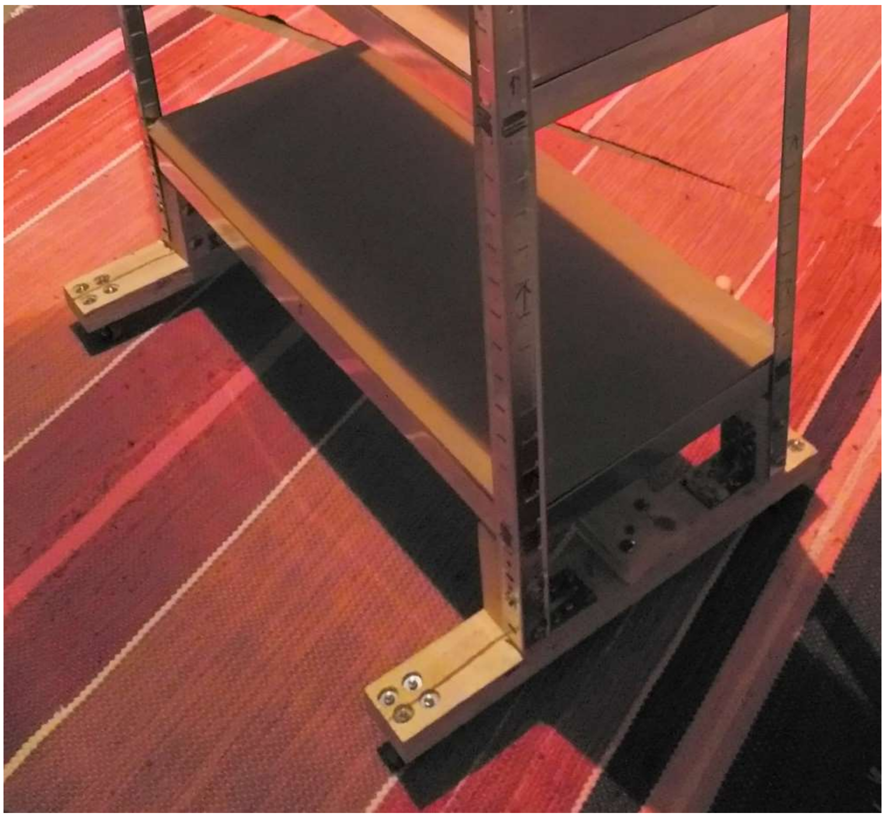


Figure 6. DIY server rack base

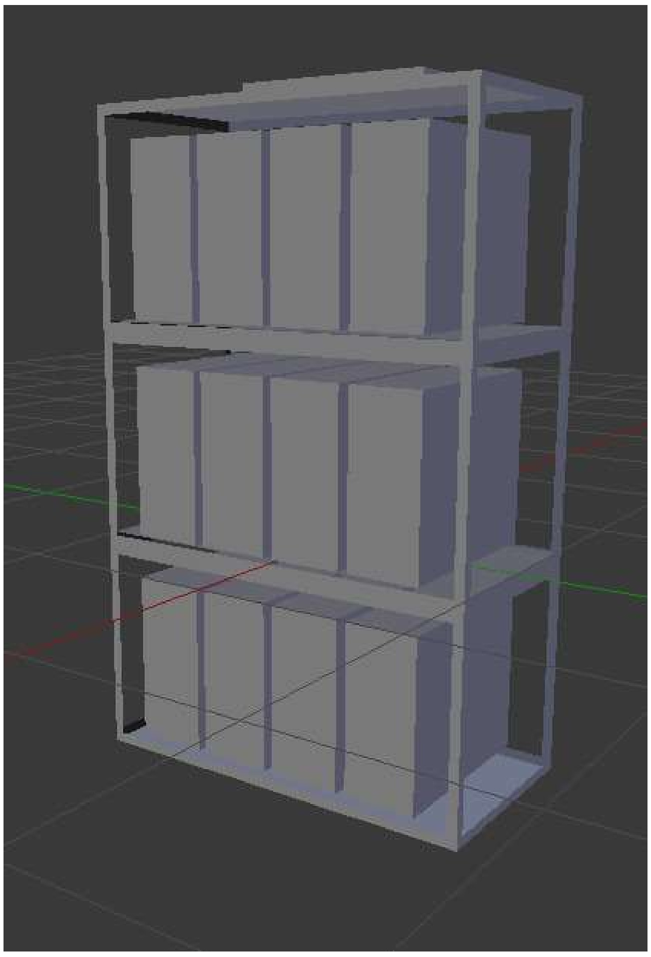


Figure 7. Original plan for the DIY server rack

As seen in Figure 7, the rack was originally designed in Blender and was meant to host maximum of 12 computers in their cases. The rack was planned after the first summer’s salvage was hauled from JAMK, and it did not count for the second summer’s salvage. After the second summer, the DIY server rack was expanded as seen in Figure 8 to hold all the 22 computers plus a 24-port network switch.

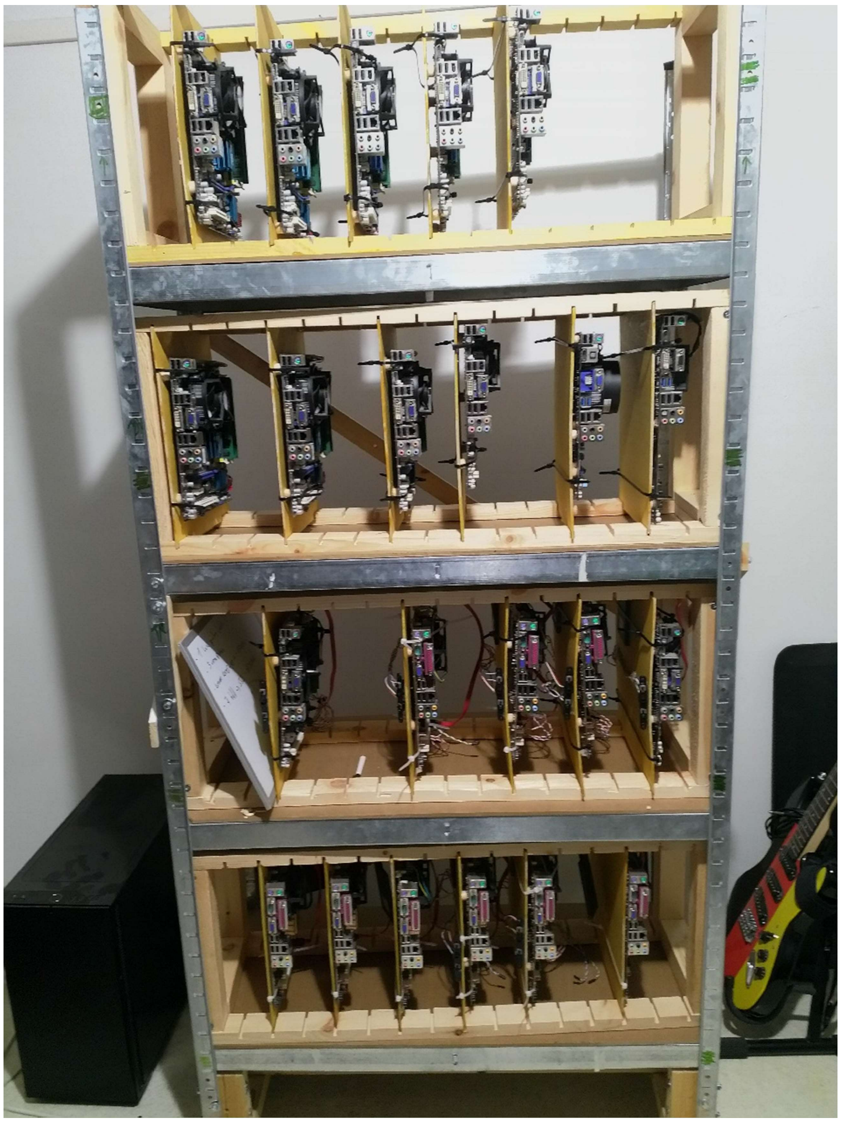


Figure 8. DIY server rack expanded

A single shelf was added and each shelf had a custom module made as shown in Figure 9 out of plank and plywood that could hold six computers motherboards.

After finishing wiring and routing the cables for all 22 computers, the result can be seen in Figure 10. Some issues arose when making the DIY rack which taught the author that if one places several motherboards with separate PSUs and wires the power button ground wrong way, turning on a single motherboard will turn on all the motherboards as described in Figure 11 also that a 10-ampere fuse is not enough to simultaneously supply power to capacitors from 22 pieces of 500 watt PSUs.

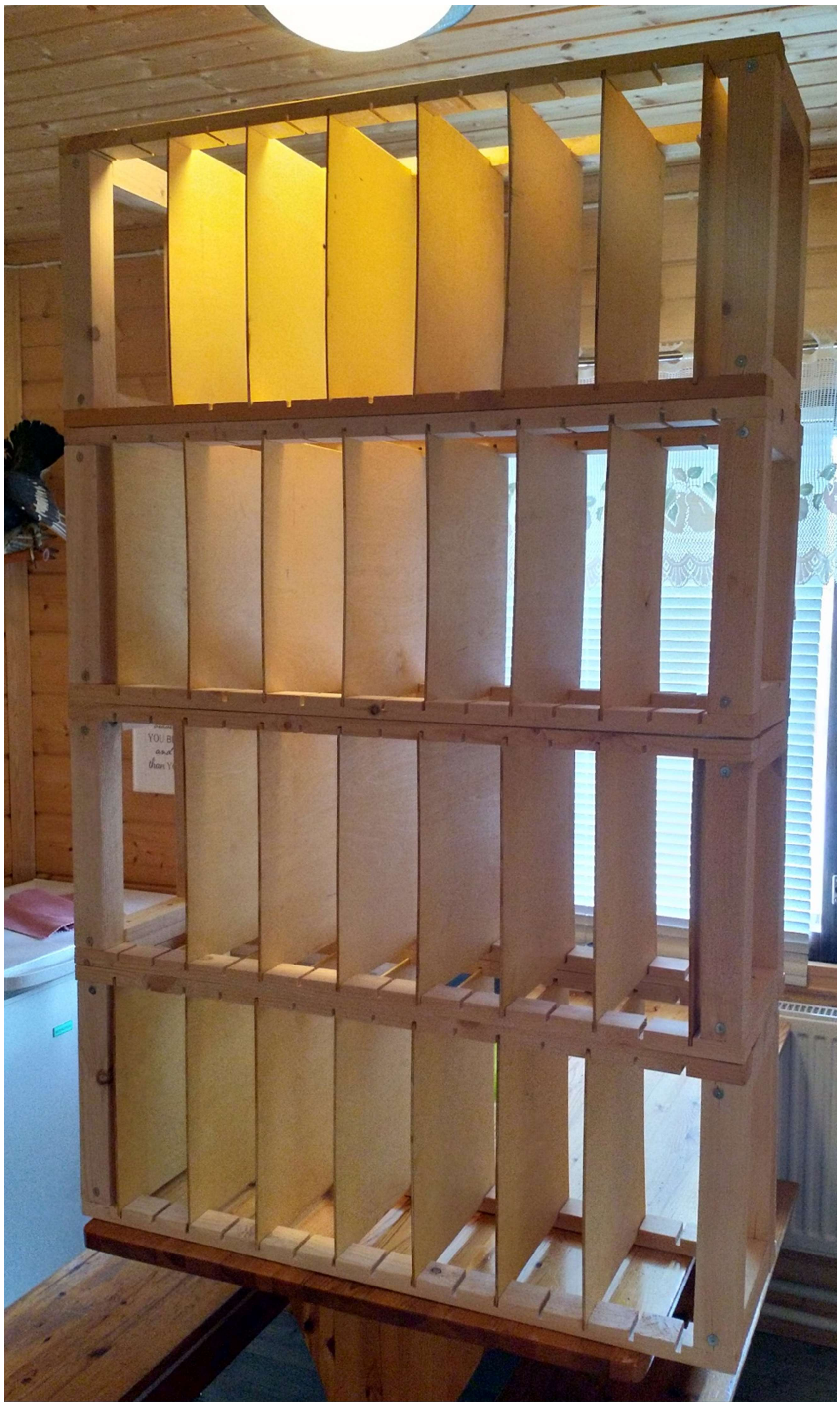


Figure 9. Motherboard modules expansion for DIY rack

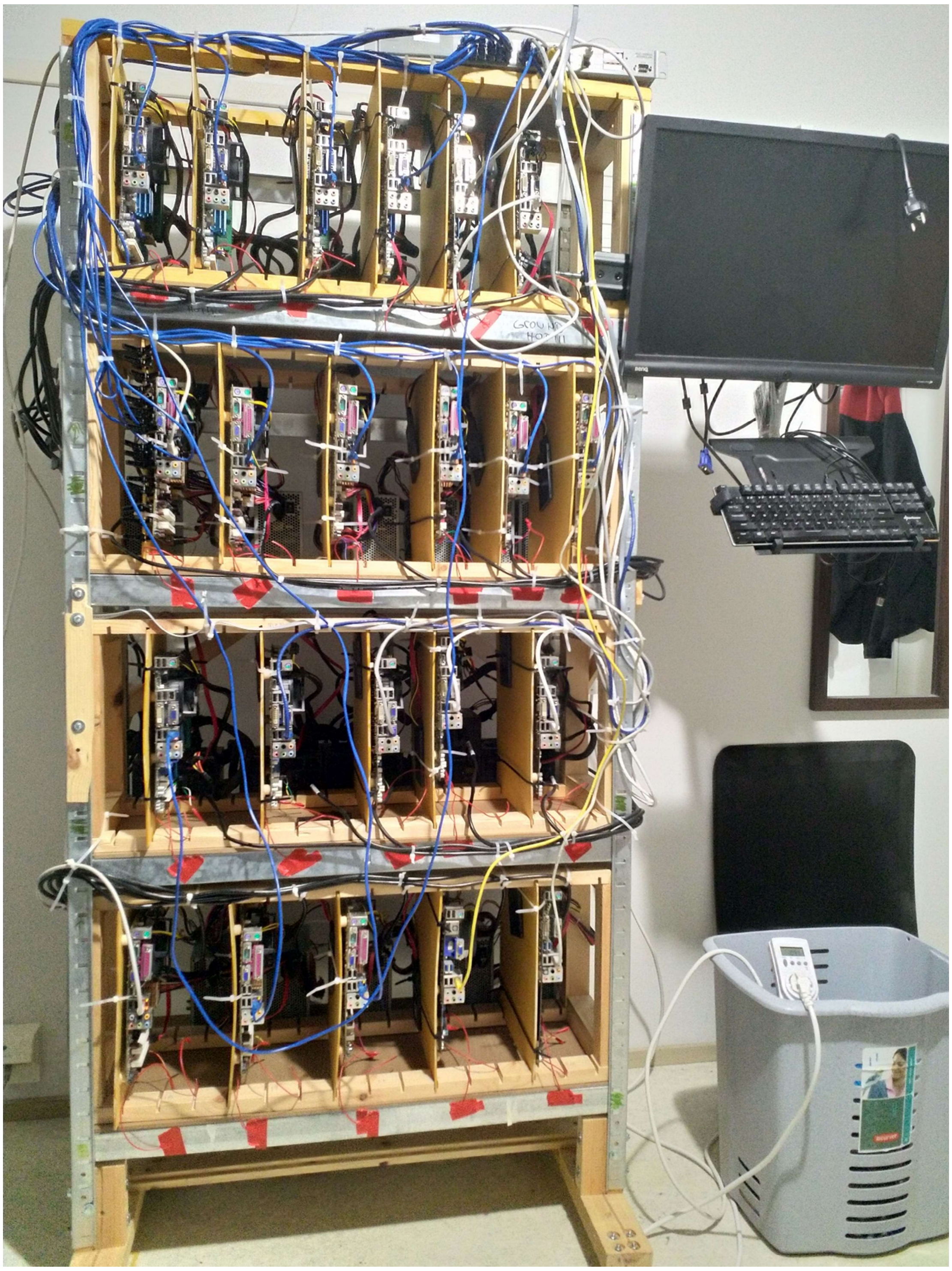


Figure 10. Finished DIY server rack

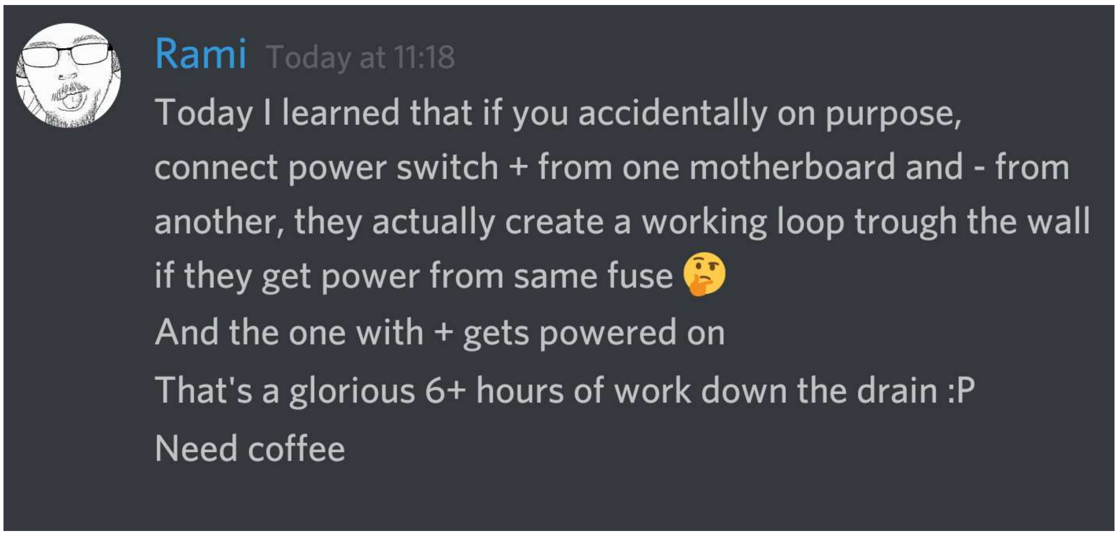

Rami Today at 11:18
Today I learned that if you accidentally on purpose, connect power switch + from one motherboard and - from another, they actually create a working loop trough the wall if they get power from same fuse 🤔
And the one with + gets powered on
That's a glorious 6+ hours of work down the drain :P
Need coffee

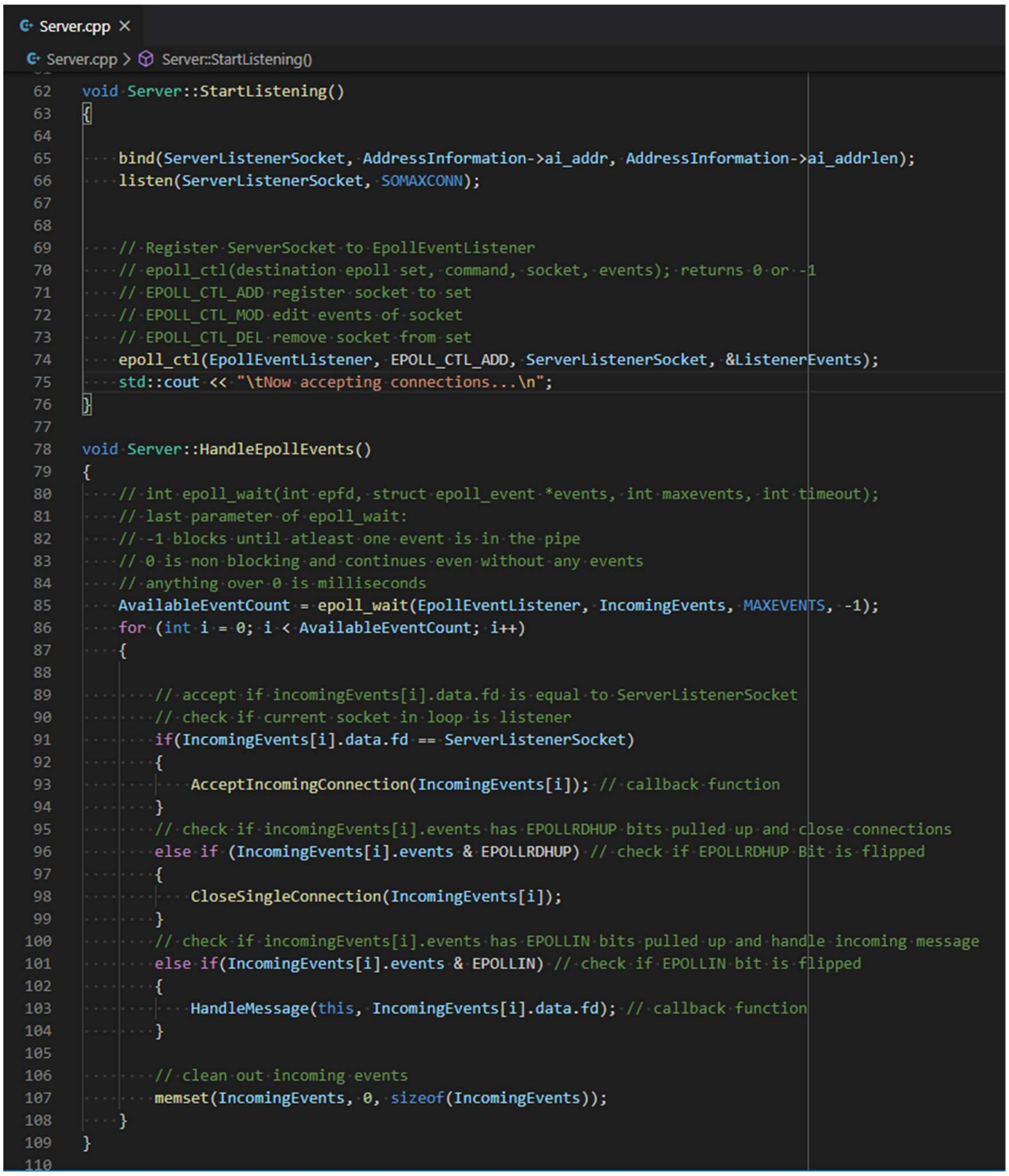
Figure 11. Accident with power rails

While making this server rack, the author had plenty of time to learn about the hardware side of hosting a multiplayer game service. The hardware was old; however, it helped the author to think what a modern system could be like. Thanks to the research into server hardware, the author was able to learn of GlusterFS and generally learn more about server architecture.

3.2.2 Chat server

One of the first pieces of software researched and developed was a prototype chat server. This server was written using C++, and the first version used Select library to asynchronously read and write sockets.

Chat server prototype was first developed in Windows using Visual Studio 2017 and Winsock library for sockets. This changed as the author researched more in to sockets and concluded that Select library does not scale up enough for a massive multiplayer game service. Select was then replaced with Epoll library, and the development was moved from Windows 10 to Ubuntu 18.04.



```
Server.cpp ×
Server.cpp > Server::StartListening()

void Server::StartListening()
{

    bind(ServerListenerSocket, AddressInformation->ai_addr, AddressInformation->ai_addrlen);
    listen(ServerListenerSocket, SOMAXCONN);


    // Register ServerSocket to EpollEventListener
    // epoll_ctl(destination epoll set, command, socket, events); returns 0 or -1
    // EPOLL_CTL_ADD register socket to set
    // EPOLL_CTL_MOD edit events of socket
    // EPOLL_CTL_DEL remove socket from set
    epoll_ctl(EpollEventListener, EPOLL_CTL_ADD, ServerListenerSocket, &ListenerEvents);
    std::cout << "\tNow accepting connections...\n";
}

void Server::HandleEpollEvents()
{
    // int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int timeout);
    // last parameter of epoll_wait:
    // -1 blocks until atleast one event is in the pipe
    // 0 is non blocking and continues even without any events
    // anything over 0 is milliseconds
    AvailableEventCount = epoll_wait(EpollEventListener, IncomingEvents, MAXEVENTS, -1);
    for (int i = 0; i < AvailableEventCount; i++)
    {

        // accept if incomingEvents[i].data.fd is equal to ServerListenerSocket
        // check if current socket in loop is listener
        if(IncomingEvents[i].data.fd == ServerListenerSocket)
        {
            AcceptIncomingConnection(IncomingEvents[i]); // callback function
        }
        // check if incomingEvents[i].events has EPOLLRDHUP bits pulled up and close connections
        else if (IncomingEvents[i].events & EPOLLRDHUP) // check if EPOLLRDHUP Bit is flipped
        {
            CloseSingleConnection(IncomingEvents[i]);
        }
        // check if incomingEvents[i].events has EPOLLIN bits pulled up and handle incoming message
        else if(IncomingEvents[i].events & EPOLLIN) // check if EPOLLIN bit is flipped
        {
            HandleMessage(this, IncomingEvents[i].data.fd); // callback function
        }

        // clean out incoming events
        memset(IncomingEvents, 0, sizeof(IncomingEvents));
    }
}
```

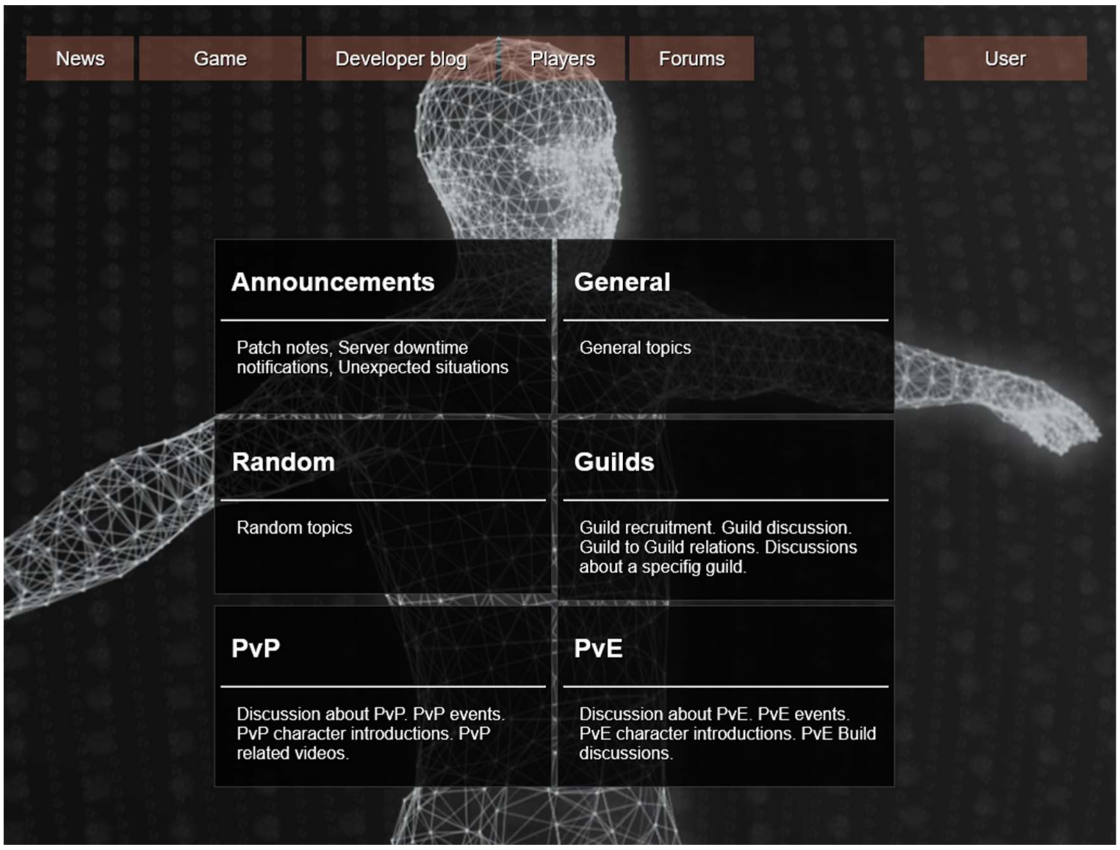
Figure 12. Epoll listener and event handling

As seen in Figure 12, the final version of the chat server prototype used Epoll C++ library on top of Linux server. While developing chat server on Ubuntu, the author started to note the similarity of a chat server to a game server and concluded that most of the code written for a simple chat server can be used to create a game server, with only the protocol changing. This discovery led to a Protocol Programming course at JAMK taught by Marko Silokunnas.

The chat prototype development was halted after hitting a major bump on the road while trying to secure the communications between server and client. The author did not have enough C++ experience and understanding on SSL/TLS, to implement either OpenSSL or LibreSSL, and the focus of the development was moved to account creation and user authentication.

3.2.3 Website

Before researching and using Django for the website, the author researched pure PHP implementation for the website, however, was quickly advised against it and was told to use Django by JAMK engineer and teacher Marko Silokunnas before any real development with PHP could be done.



Announcements

Patch notes, Server downtime notifications, Unexpected situations

General

General topics

Random

Random topics

Guilds

Guild recruitment. Guild discussion. Guild to Guild relations. Discussions about a specifig guild.

PvP

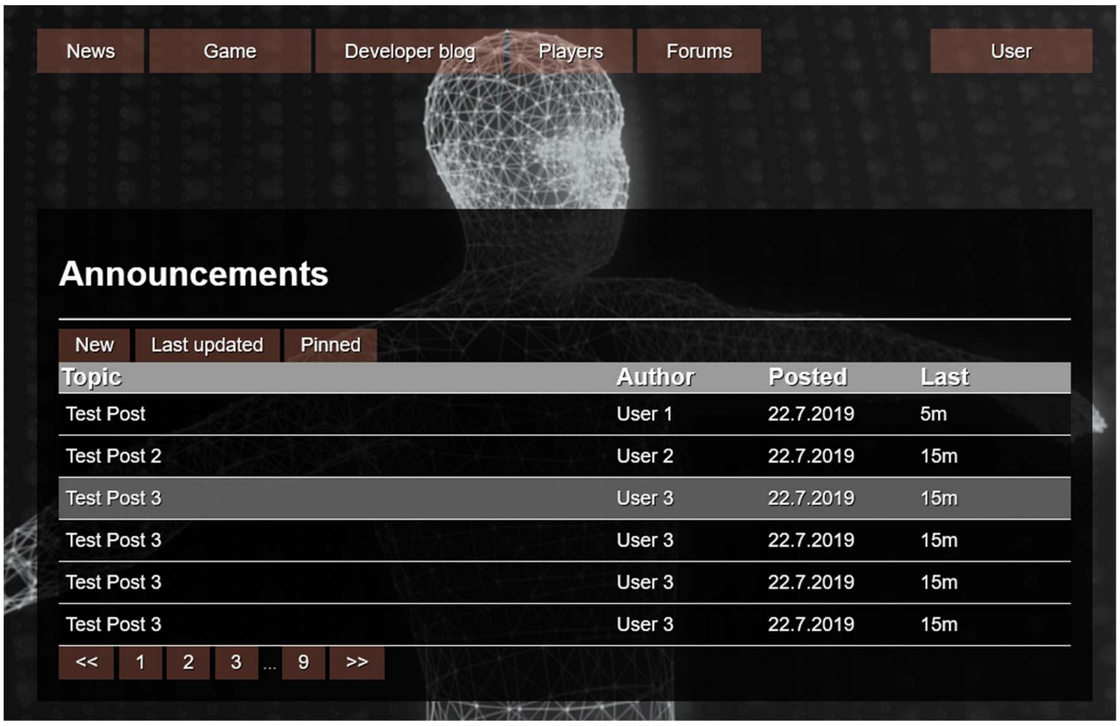
Discussion about PvP. PvP events. PvP character introductions. PvP related videos.

PvE

Discussion about PvE. PvE events. PvE character introductions. PvE Build discussions.

Figure 13. Forums module for Django

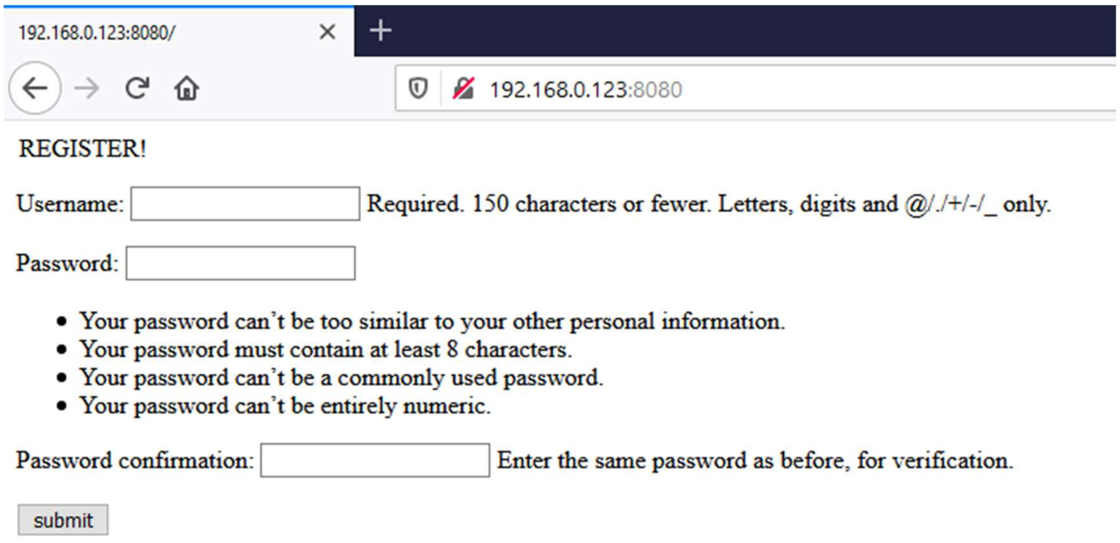
The first version of Django website was quickly up with the help of stack overflow and YouTube tutorials by Corey Schafer (Schafer 2019.) and some time was spent to develop a website template and news and forums modules for Django as seen in Figure 13. These modules were mostly stylized as seen in Figure 14; however, they never had any backend code implemented for them, and the author felt that this was not part of the multiplayer service components per se; hence, the styling and module development were left to a minimum and later removed completely from the account creation website used in the login server.



Topic	Author	Posted	Last
Test Post	User 1	22.7.2019	5m
Test Post 2	User 2	22.7.2019	15m
Test Post 3	User 3	22.7.2019	15m
Test Post 3	User 3	22.7.2019	15m
Test Post 3	User 3	22.7.2019	15m
Test Post 3	User 3	22.7.2019	15m

Figure 14. Announcements topics posts

After getting a basic account register website up as seen in Figure 15, most of the time was spent on researching MariaDB connection with Django. MariaDB was chosen as the database for the website and for the game service, and the biggest hurdle was to get Django to communicate with MariaDB. Finally, the author was able to make Django communicate with MariaDB by installing a MySQL driver called "mysqlclient-1.4.4-cp38-cp38-win32.whl". This MySQL client was installed inside Python virtualenv using wheel.


192.168.0.123:8080/

192.168.0.123:8080

REGISTER!

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

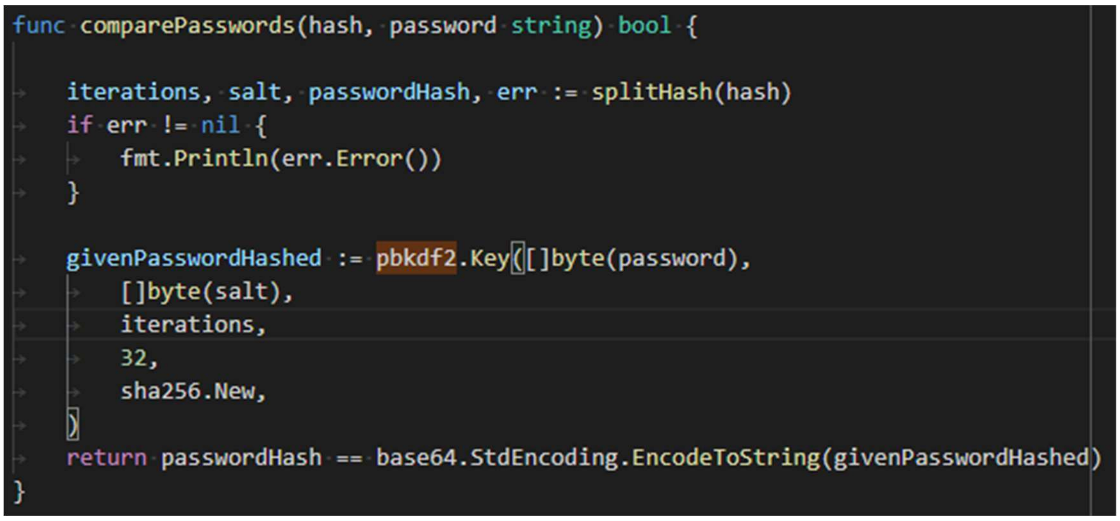
submit

Figure 15. Basic account registration website

The author was slightly worried about using Django, as the user module was using a password hashing algorithm called "PBKDF2" which was unknown before this. The research into C++ library able to handle the algorithm led back to OpenSSL and LibreSSL; however, research into these technologies before did not yield very good results; hence, the author started to fall back to Google's Go language and the research into Go's security libraries yielded very good results. Because of this research into Go's security libraries, the author after discussions with Marko Silokunnas concerning the performance differences between Go and C++ being negligent started to fall back to Go on all development.

3.2.4 Login server

User authentication server was developed using Go, and it was the second software component researched and developed by the author. Security libraries played a big part for selecting Go as the programming language for authentication. Go's security libraries allowed several days' worth of development and research in C++ be replaced with a single day's worth of research and development.



```go
func comparePasswords(hash, password string) bool {

    iterations, salt, passwordHash, err := splitHash(hash)
    if err != nil {
        fmt.Println(err.Error())
    }

    givenPasswordHashed := pbkdf2.Key([]byte(password),
        []byte(salt),
        iterations,
        32,
        sha256.New,
    )
    return passwordHash == base64.StdEncoding.EncodeToString(givenPasswordHashed)
}
```

Figure 16. Pbkdf2 algorithm in use

After getting properly salted and hashed password from the database, it was split into three parts. Iterations, Salt and Hash. Iterations and salt are used on the client's password and that result is compared against the Hash retrieved from the database, returning either true or false as seen in Figure 16. Because of this one simple function, the author decided to move back from C++ to Go.

The major issues while designing the login server were that the author had not done anything like it before and was starting from scratch. The author designed a flow chart as seen in Figure 17 over multiple iterations. When designing the authentication flow chart, the author thought of a single log in attempt and the path it takes, first generally thinking the steps, then opening those steps further, which eventually led to a flow chart in Figure 17.

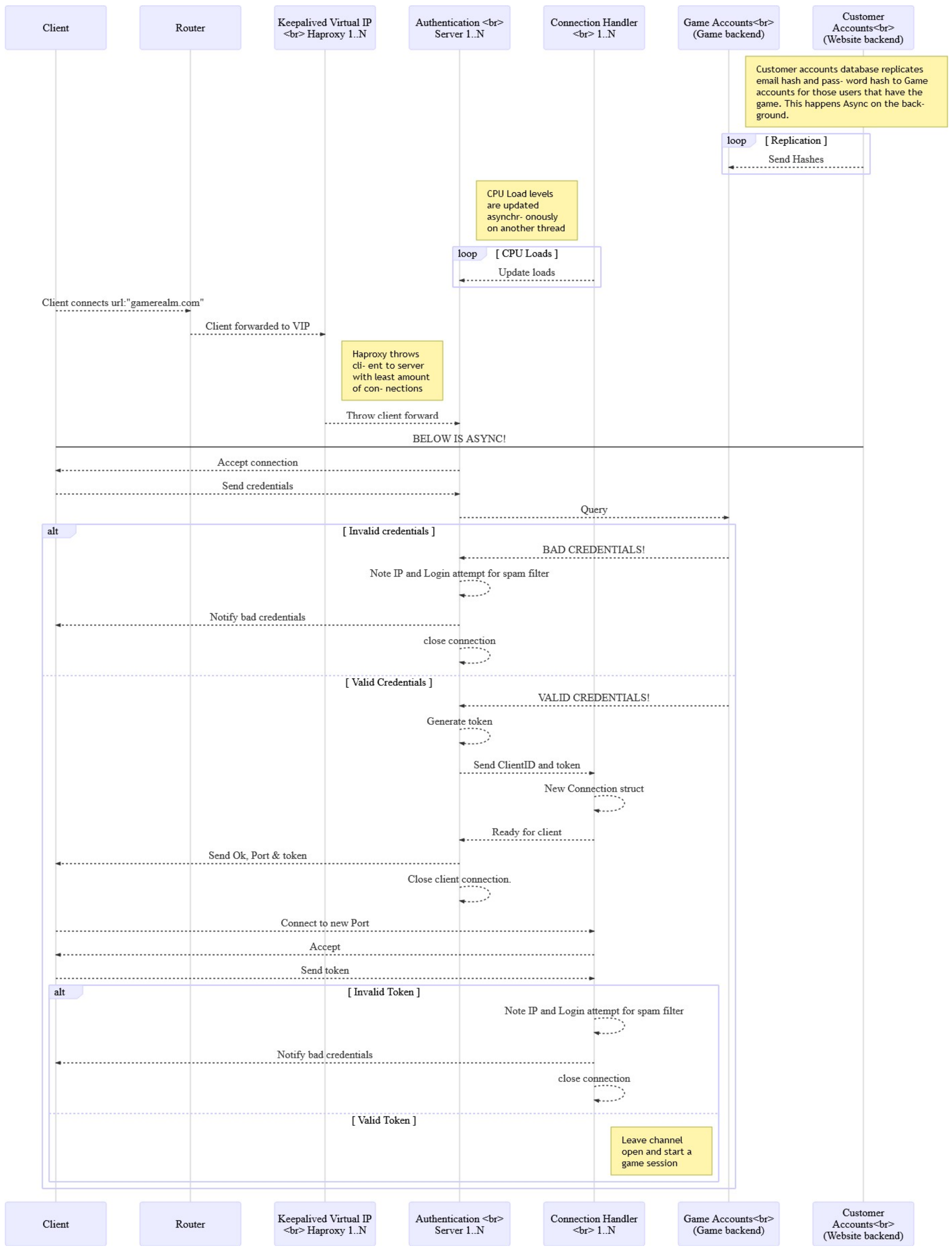
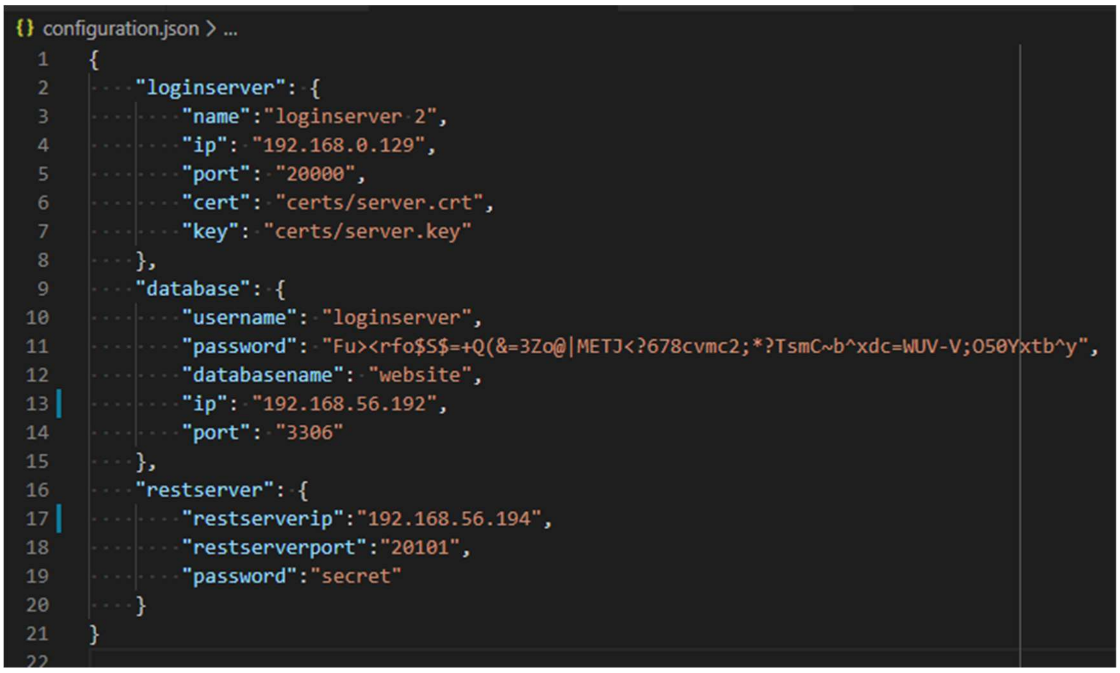

Client
Router
Keepalived Virtual IP
 Haproxy 1..N
Authentication
 Server 1..N
Connection Handler
 1..N
Game Accounts
 (Game backend)
Customer Accounts
 (Website backend)
Customer accounts database replicates email hash and pass- word hash to Game accounts for those users that have the game. This happens Async on the back- ground.
loop
[Replication]
Send Hashes
CPU Load levels are updated asynchr- onously on another thread
loop
[CPU Loads]
Update loads
Client connects url:"gamerealm.com"
Client forwarded to VIP
Haproxy throws cli- ent to server with least amount of con- nections
Throw client forward
BELOW IS ASYNC!
Accept connection
Send credentials
Query
alt
[Invalid credentials]
BAD CREDENTIALS!
Note IP and Login attempt for spam filter
Notify bad credentials
close connection
[Valid Credentials]
VALID CREDENTIALS!
Generate token
Send ClientID and token
New Connection struct
Ready for client
Send Ok, Port & token
Close client connection.
Connect to new Port
Accept
Send token
alt
[Invalid Token]
Note IP and Login attempt for spam filter
Notify bad credentials
close connection
[Valid Token]
Leave channel open and start a game session

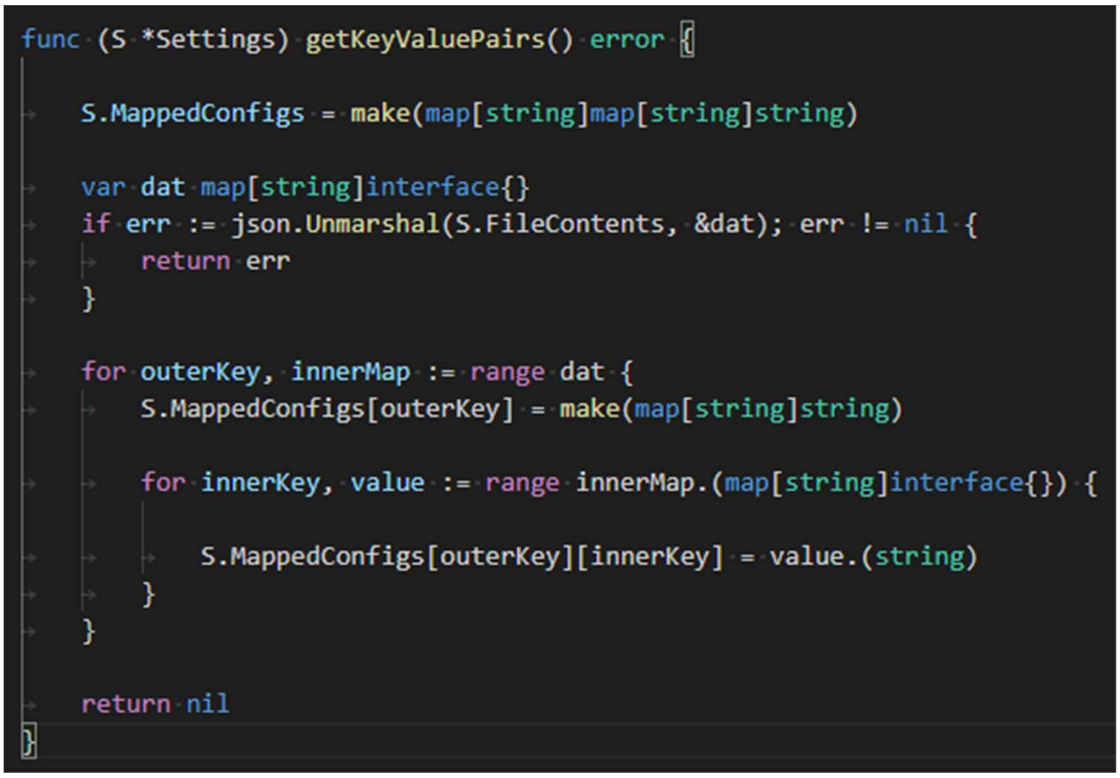
Figure 17. Authentication flow chart

As seen in Figure 18, all the settings are stored in a JSON file as key value pairs. These key value pairs are parsed from the JSON file using a custom code seen in Figure 19. Each Key is stored and called from a hash map, by the program using the configurations getter code.



```json
{} configuration.json > ...
{
    "loginserver": {
        "name":"loginserver 2",
        "ip": "192.168.0.129",
        "port": "20000",
        "cert": "certs/server.crt",
        "key": "certs/server.key"
    },
    "database": {
        "username": "loginserver",
        "password": "Fu><rfo$S$=+Q(&=3Zo@|METJ<?678cvmc2;*?TsmC~b^xdc=WUV-V;O50Yxtb^y",
        "databasename": "website",
        "ip": "192.168.56.192",
        "port": "3306"
    },
    "restserver": {
        "restserverip":"192.168.56.194",
        "restserverport":"20101",
        "password":"secret"
    }
}
```

Figure 18. JSON file storing settings



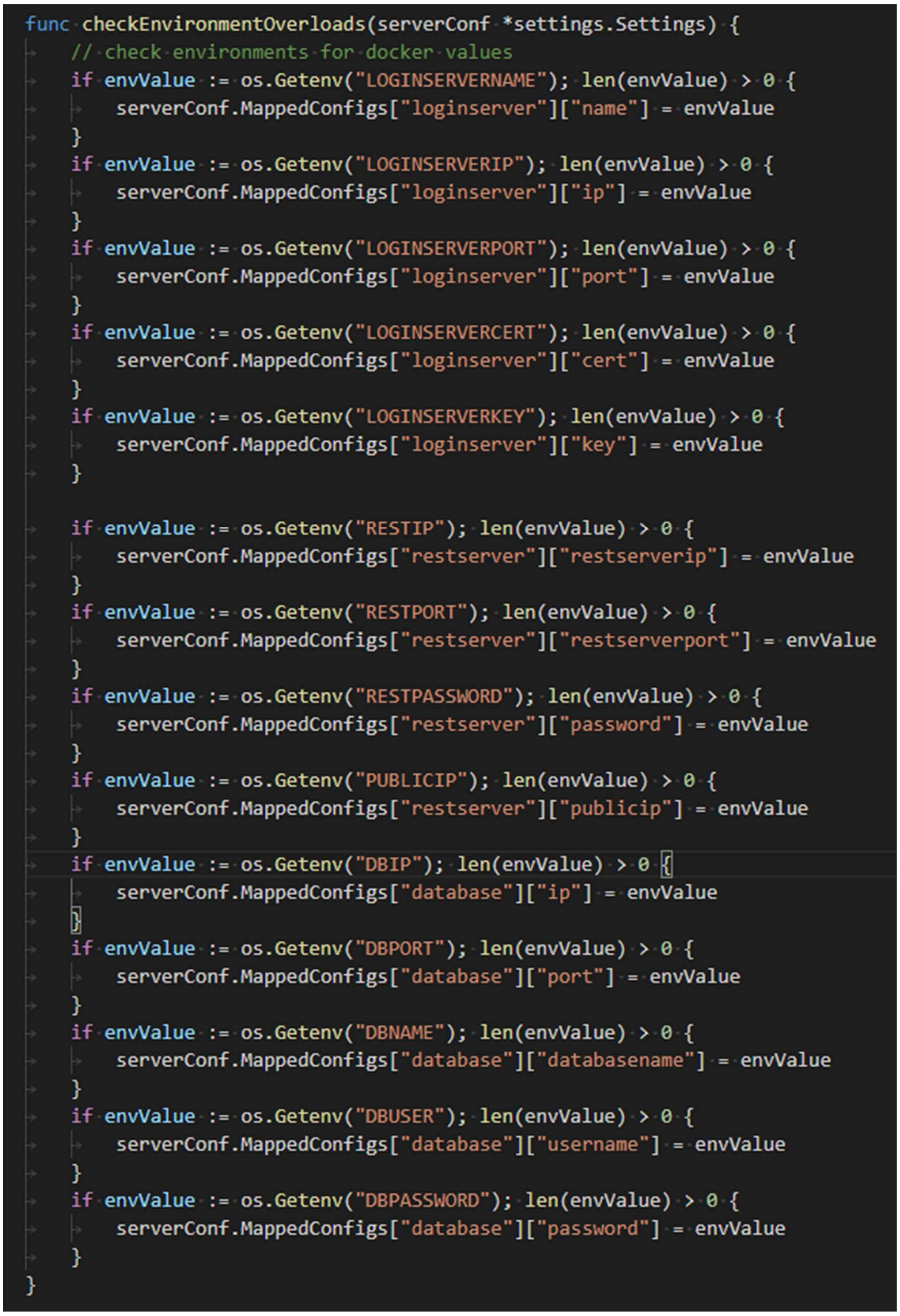
```go
func (S *Settings) getKeyValuePairs() error {

    S.MappedConfigs = make(map[string]map[string]string)

    var dat map[string]interface{}
    if err := json.Unmarshal(S.FileContents, &dat); err != nil {
        return err
    }

    for outerKey, innerMap := range dat {
        S.MappedConfigs[outerKey] = make(map[string]string)

        for innerKey, value := range innerMap.(map[string]interface{}) {

            S.MappedConfigs[outerKey][innerKey] = value.(string)
        }
    }

    return nil
}
```

Figure 19. JSON settings parser

The usefulness for this style of settings management is that it makes implementing new settings easy. A Developer can insert a new key value pair to the configuration file and just call the key in code for it to work; it also consolidates all the program settings to a JSON file, easily recognizable by fellow developers.

Docker was used in the login server in such a way that creating a new image from the Dockerfile and running it as a container, would always run the latest version of login server available. This means that the Dockerfile when built, sets up an Alpine environment for building and running Go applications, gets login server project from non-public Git over SSH, builds it, then runs it. A trained system administrator would quickly notice an issue here. The configuration file is stored inside the Git project, so automating the component deployment becomes impossible, unless there is a way to pass the settings to the container. This problem caused all the earlier usefulness with the configuration getter code to disappear, since basically any user given variable that is stored inside the project was out of the question.

The author did not spend much time on researching this new settings problem that rose up during the last meters of the development, instead the author “hot fixed” it by simply adding couple of lines of code, as seen in the Figure 20. The author added an overload function that would check Operating system level predefined environment variables and replace the settings in configuration file. This was a “quick and dirty” solution for a problem that arose when trying to use Vagrant together with login server Dockerfile to create a single PC test environment script during Automation of Services course held by Jarmo Viinikanoja.



```go
func checkEnvironmentOverloads(serverConf *settings.Settings) {
    // check environments for docker values
    if envValue := os.Getenv("LOGINSERVERNAME"); len(envValue) > 0 {
        serverConf.MappedConfigs["loginserver"]["name"] = envValue
    }
    if envValue := os.Getenv("LOGINSERVERIP"); len(envValue) > 0 {
        serverConf.MappedConfigs["loginserver"]["ip"] = envValue
    }
    if envValue := os.Getenv("LOGINSERVERPORT"); len(envValue) > 0 {
        serverConf.MappedConfigs["loginserver"]["port"] = envValue
    }
    if envValue := os.Getenv("LOGINSERVERCERT"); len(envValue) > 0 {
        serverConf.MappedConfigs["loginserver"]["cert"] = envValue
    }
    if envValue := os.Getenv("LOGINSERVERKEY"); len(envValue) > 0 {
        serverConf.MappedConfigs["loginserver"]["key"] = envValue
    }

    if envValue := os.Getenv("RESTIP"); len(envValue) > 0 {
        serverConf.MappedConfigs["restserver"]["restserverip"] = envValue
    }
    if envValue := os.Getenv("RESTPORT"); len(envValue) > 0 {
        serverConf.MappedConfigs["restserver"]["restserverport"] = envValue
    }
    if envValue := os.Getenv("RESTPASSWORD"); len(envValue) > 0 {
        serverConf.MappedConfigs["restserver"]["password"] = envValue
    }
    if envValue := os.Getenv("PUBLICIP"); len(envValue) > 0 {
        serverConf.MappedConfigs["restserver"]["publicip"] = envValue
    }
    if envValue := os.Getenv("DBIP"); len(envValue) > 0 {
        serverConf.MappedConfigs["database"]["ip"] = envValue
    }
    if envValue := os.Getenv("DBPORT"); len(envValue) > 0 {
        serverConf.MappedConfigs["database"]["port"] = envValue
    }
    if envValue := os.Getenv("DBNAME"); len(envValue) > 0 {
        serverConf.MappedConfigs["database"]["databasename"] = envValue
    }
    if envValue := os.Getenv("DBUSER"); len(envValue) > 0 {
        serverConf.MappedConfigs["database"]["username"] = envValue
    }
    if envValue := os.Getenv("DBPASSWORD"); len(envValue) > 0 {
        serverConf.MappedConfigs["database"]["password"] = envValue
    }
}
```

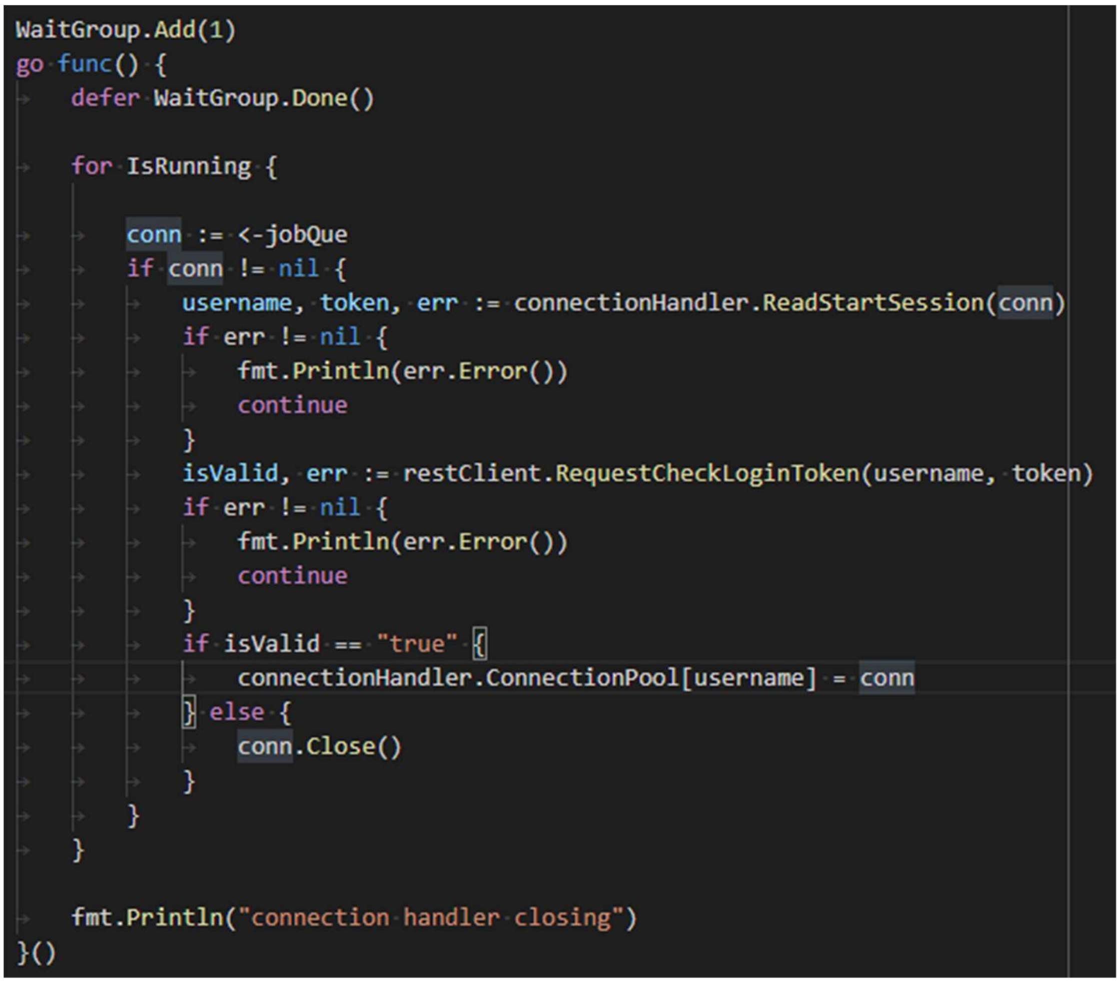
Figure 20. Environment variables overloads

Login protocol was also created as a written document to help develop a client to server connection possibly written in different programming languages, so they could communicate together properly. This protocol can be read fully in this document in Appendix 1.

3.2.5 Connection handler stub

This piece of programming is not really that big; yet, it is important because it highlighted to the author that a REST server is necessary for this type of a system. The initial idea was to have login server and connection handler to communicate through a secondary socket; however, as the scalability was an important factor for this thesis, it was quickly abandoned.

Connection handler as a component in the system was tasked to pool client connections and work as a relay between client and a world server. This would help to create a small barrier between a world server and a possible DoS attack, and possibly moving some of the calculations from the world server to the connection handler.



```go
WaitGroup.Add(1)
go func() {
    defer WaitGroup.Done()

    for IsRunning {

        conn := <-jobQue
        if conn != nil {
            username, token, err := connectionHandler.ReadStartSession(conn)
            if err != nil {
                fmt.Println(err.Error())
                continue
            }
            isValid, err := restClient.RequestCheckLoginToken(username, token)
            if err != nil {
                fmt.Println(err.Error())
                continue
            }
            if isValid == "true" {
                connectionHandler.ConnectionPool[username] = conn
            } else {
                conn.Close()
            }
        }
    }

    fmt.Println("connection handler closing")
}()
```

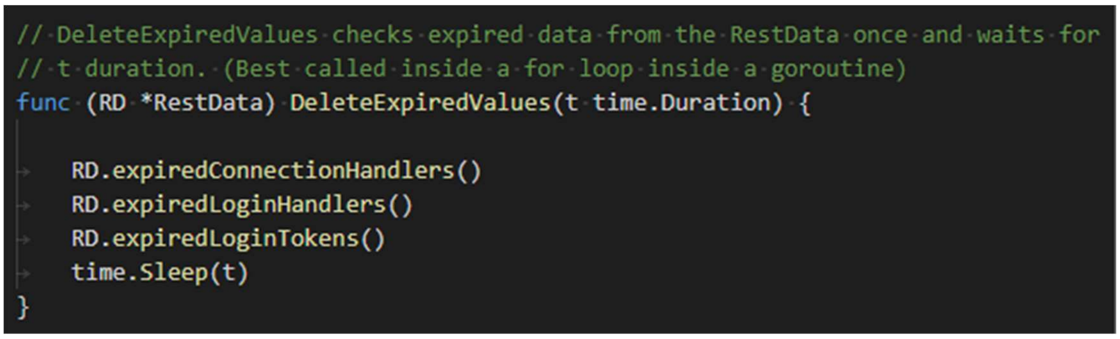
Figure 21. Connection handler worker loop

The final version of the stub as seen in Figure 21 would not return anything for the client; instead, it would print out an error line in cases where a client’s session token did not match the one in REST server.

3.2.6 Rest server

REST server was added after the need for it was recognized while implementing the connection handler stub. It became quickly apparent that when scaling the system upwards, keeping track of all the IP addresses and what component runs on which address was going to become a problem; hence, it was decided that a REST server would be implemented and each component would register its information to the REST. This way the only address the system components needed to know was REST server address and get other information through that.

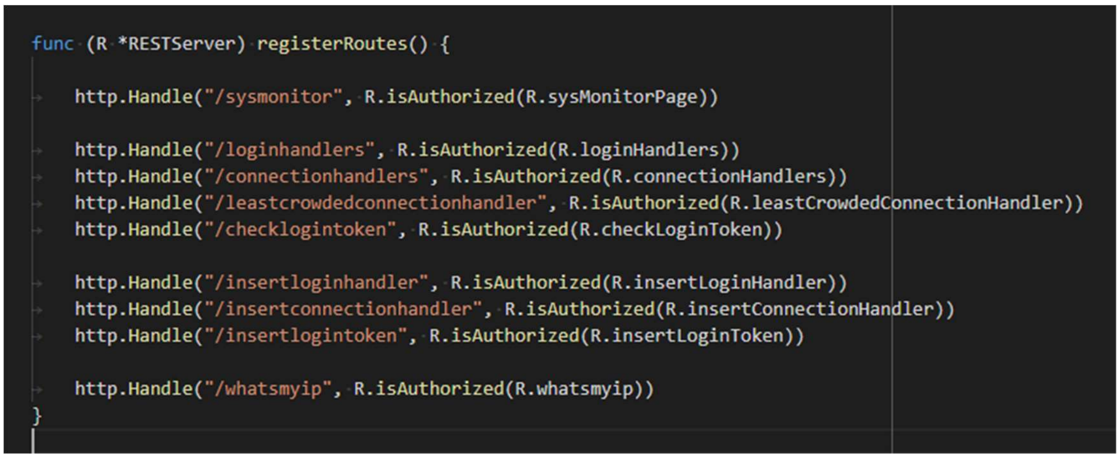
REST server was later extended to share other data as well, for example, the login server generates a session token for client that is sent to REST and to the client, which the client then uses to authenticate itself to the connection handler that checks it from REST again.



```go
// DeleteExpiredValues checks expired data from the RestData once and waits for
// t duration. (Best called inside a for loop inside a goroutine)
func (RD *RestData) DeleteExpiredValues(t time.Duration) {

	RD.expiredConnectionHandlers()
	RD.expiredLoginHandlers()
	RD.expiredLoginTokens()
	time.Sleep(t)
}
```

Figure 22. DeleteExpiredValues goroutine

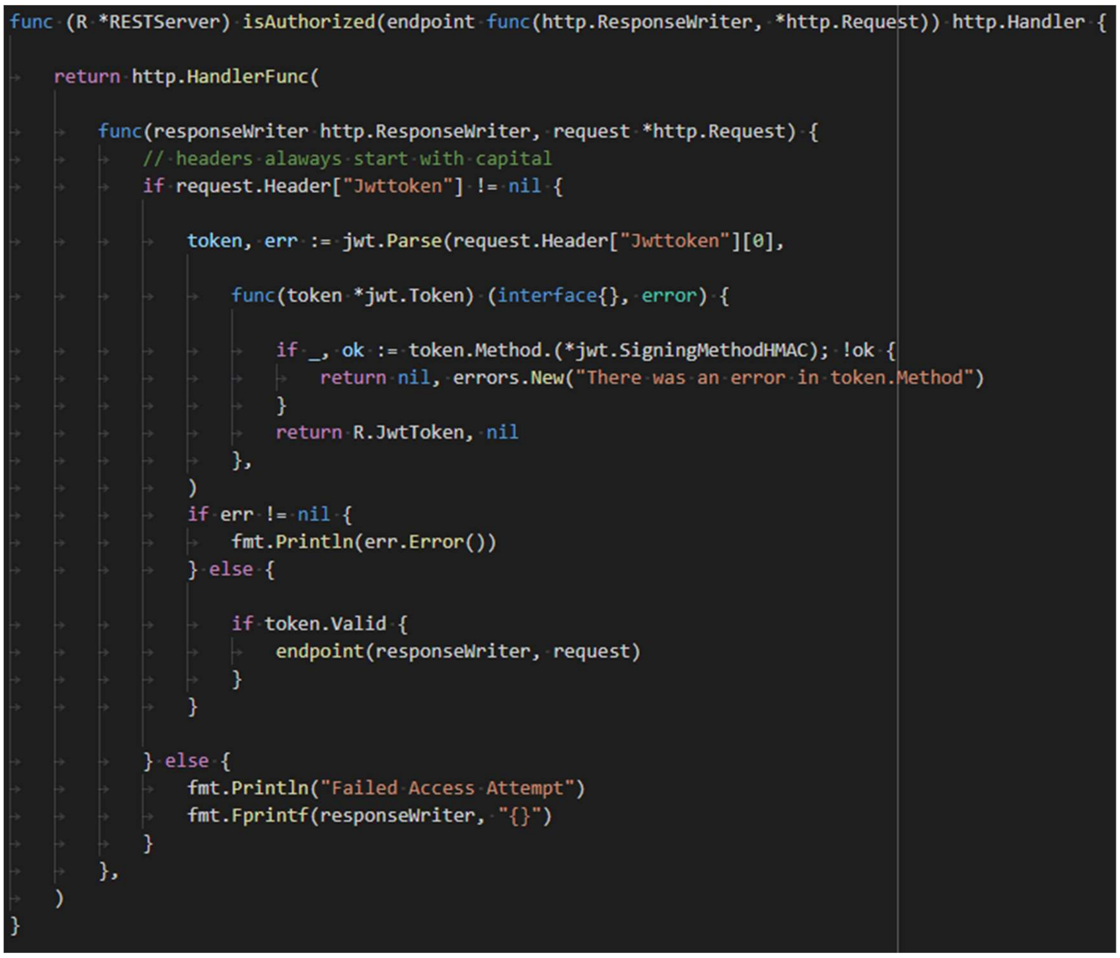
To host session token data, Redis server cluster was researched as a storage medium because Redis values could have lifetimes which would be perfect for short term session key data that should not exist more than 5 seconds. It would also enable REST server to be scalable sideways, which it is currently not able to do. Due to some difficulties creating a Redis cluster and running out of time, simple REST data code was programmed, which simulated Redis lifetime. Each component in the system must update its information in the REST server, or it will be removed by a goroutine as seen in Figure 22. This could also be used as a system monitor hook that monitors REST data and alerts if a component does not update itself.



```
func (R *RESTServer) registerRoutes() {

    http.Handle("/sysmonitor", R.isAuthorized(R.sysMonitorPage))

    http.Handle("/loginhandlers", R.isAuthorized(R.loginHandlers))
    http.Handle("/connectionhandlers", R.isAuthorized(R.connectionHandlers))
    http.Handle("/leastcrowdedconnectionhandler", R.isAuthorized(R.leastCrowdedConnectionHandler))
    http.Handle("/checklogintoken", R.isAuthorized(R.checkLoginToken))

    http.Handle("/insertloginhandler", R.isAuthorized(R.insertLoginHandler))
    http.Handle("/insertconnectionhandler", R.isAuthorized(R.insertConnectionHandler))
    http.Handle("/insertlogintoken", R.isAuthorized(R.insertLoginToken))

    http.Handle("/whatsmyip", R.isAuthorized(R.whatsmyip))
}
```

Figure 23. REST server routes

The REST component was developed with Go and MVC, and as seen in Figure 23, it had routes linking them to functions that suited the route specifically. System monitor REST route would render an HTTP page with all the REST data it currently had.



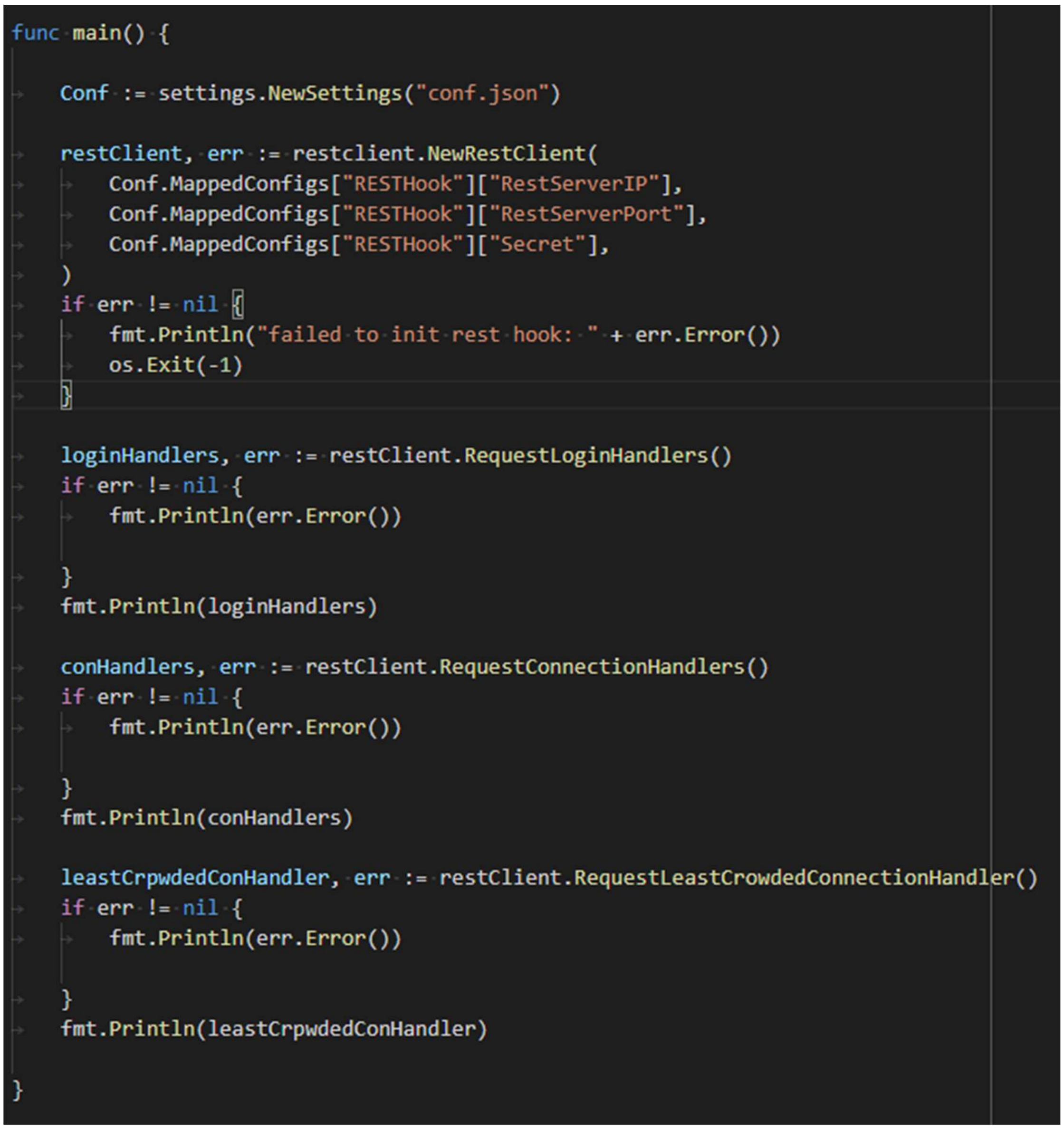
```
func (R *RESTServer) isAuthorized(endpoint func(http.ResponseWriter, *http.Request)) http.Handler {

    return http.HandlerFunc(

        func(responseWriter http.ResponseWriter, request *http.Request) {
            // headers alaways start with capital
            if request.Header["Jwttoken"] != nil {

                token, err := jwt.Parse(request.Header["Jwttoken"][0],

                    func(token *jwt.Token) (interface{}, error) {

                        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
                            return nil, errors.New("There was an error in token.Method")
                        }
                        return R.JwtToken, nil
                    },
                )
                if err != nil {
                    fmt.Println(err.Error())
                } else {

                    if token.Valid {
                        endpoint(responseWriter, request)
                    }
                }

            } else {
                fmt.Println("Failed Access Attempt")
                fmt.Fprintf(responseWriter, "{}")
            }
        },
    )
}
```

Figure 24. JWT REST server authentication

REST server was secured using TLS certifications and required a password used by a JWT token library that would prevent unauthorized access to REST data as seen in Figure 24. In case of a faulty password, REST server would return only basic brackets and leave out every piece of information a hacker could use to deduce something.

3.2.7 Rest test client

As seen in Figure 25, a simple REST client program was created to query data and debug the REST server and REST client code. This client simply used the REST client code to build an HTTP packet with proper headers and sent it to the proper REST server route.



```go
func main() {

    Conf := settings.NewSettings("conf.json")

    restClient, err := restclient.NewRestClient(
        Conf.MappedConfigs["RESTHook"]["RestServerIP"],
        Conf.MappedConfigs["RESTHook"]["RestServerPort"],
        Conf.MappedConfigs["RESTHook"]["Secret"],
    )
    if err != nil {
        fmt.Println("failed to init rest hook: " + err.Error())
        os.Exit(-1)
    }

    loginHandlers, err := restClient.RequestLoginHandlers()
    if err != nil {
        fmt.Println(err.Error())

    }
    fmt.Println(loginHandlers)

    conHandlers, err := restClient.RequestConnectionHandlers()
    if err != nil {
        fmt.Println(err.Error())

    }
    fmt.Println(conHandlers)

    leastCrpwdedConHandler, err := restClient.RequestLeastCrowdedConnectionHandler()
    if err != nil {
        fmt.Println(err.Error())

    }
    fmt.Println(leastCrpwdedConHandler)

}
```

Figure 25. REST test client

3.2.8 Unity test client

To test the authentication pipeline with registration and login, a test client was created using Unity 3D game engine. This was also the first test for the login protocol the author wrote for bridging a Go language-based login server and a Mono-C# based game client.

The game test client was only a basic empty scene with OnGUI scripts attached to the camera as can be seen in Figure 26. OnGUI is Unity’s interface function that the engine calls when drawing UI layer.

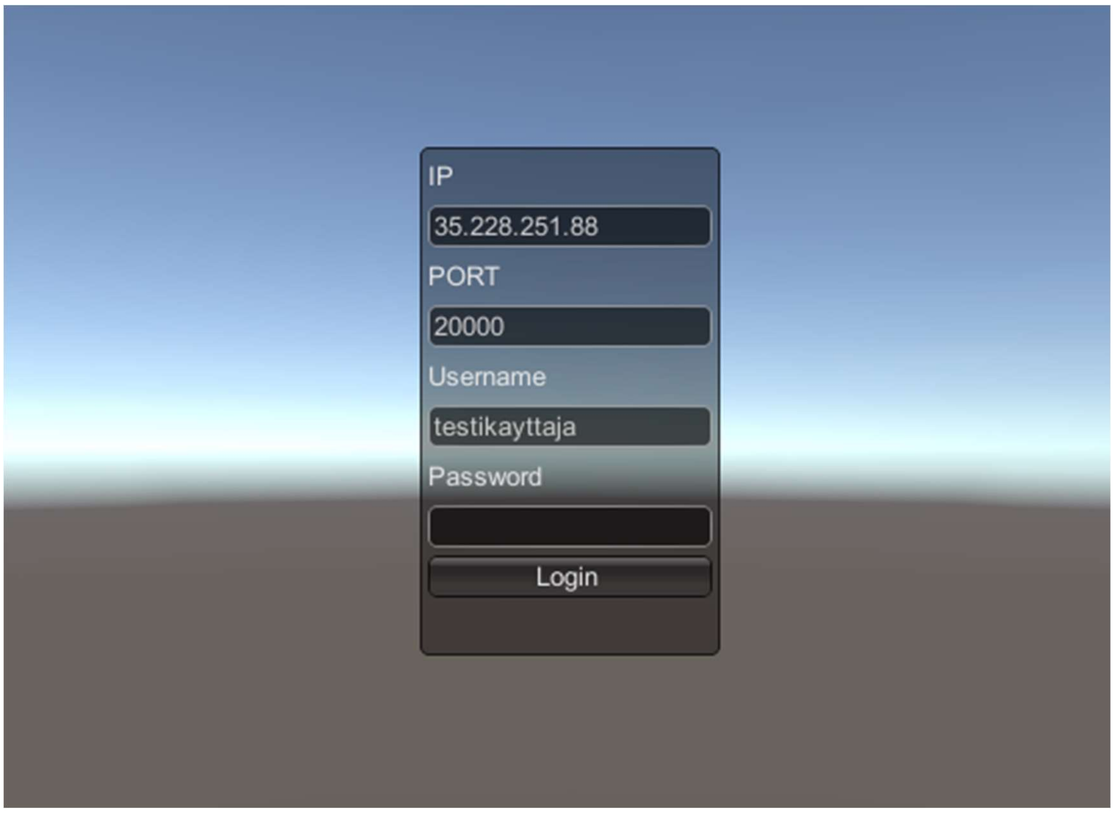
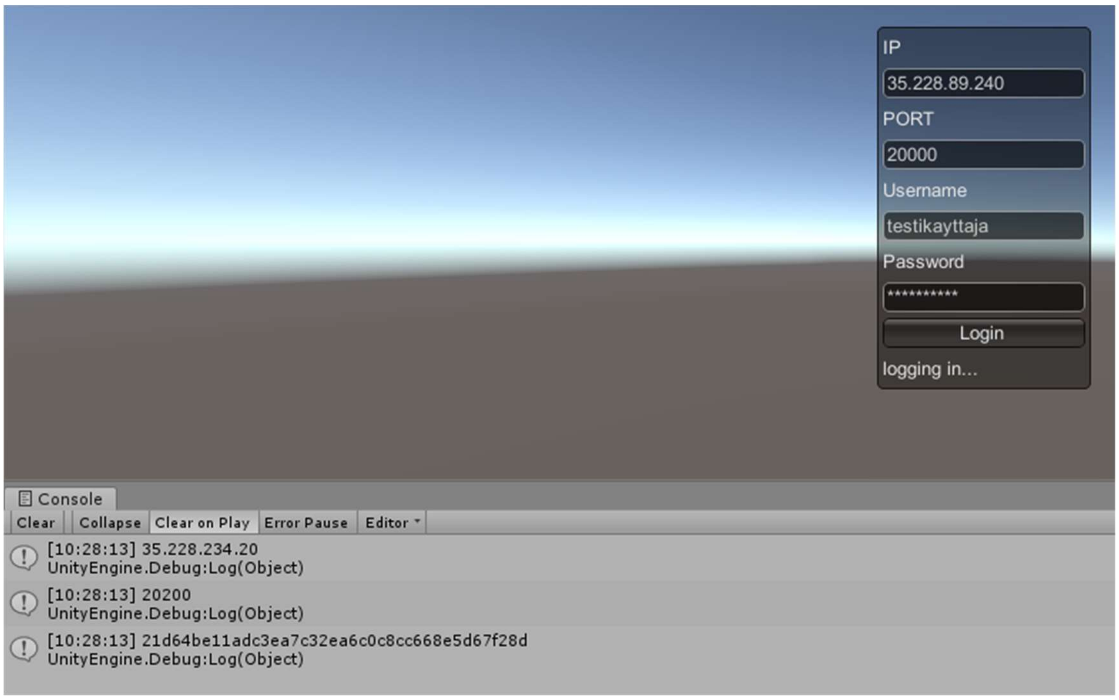


Figure 26. Unity authentication pipe test client

When pressing the login button, the script would take the username and password and add them to a binary stream specified by the login protocol (described in Appendix 1), after which the script would attempt to connect to a TLS socket in the specified IP address. Thanks to the protocol created, the communication between Mono C# and Go programming language was successful. After successfully authenticating the client, the login server sends client a session token and checks the least crowded connection handler and its port to the player, which can be seen in Figure 27, where the game client writes the port number and hash token to console.



logging in...

```
[10:28:13] 35.228.234.20
UnityEngine.Debug:Log(Object)
[10:28:13] 20200
UnityEngine.Debug:Log(Object)
[10:28:13] 21d64be11adc3ea7c32ea6c0c8cc668e5d67f28d
UnityEngine.Debug:Log(Object)
```

Figure 27. Successful connection from game client to login server

3.2.9 MariaDB Galera Cluster + GlusterFS

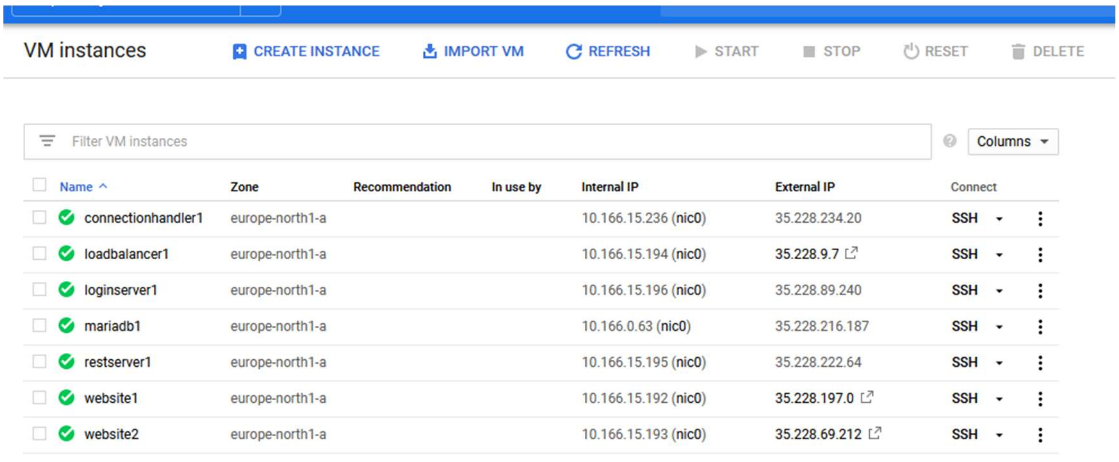
MariaDB Galera Cluster was created as a test; however, no real development took place on top of it. The author was able to achieve 3 node MariaDB Galera Cluster on top of the virtual machines. No actual development data went to this database, as Docker containerizing MariaDB with Galera Cluster was impossible without GlusterFS network share.

GlusterFS was researched and implemented on top of virtual machines after implementing Galera Cluster to other virtual machines as a part of attempting to containerize MariaDB galera cluster with Docker. The author did not manage to do this in timely manner, and the author decided to postpone this feature after Automation of Services course.

3.2.10 System automations

At the final stretch of the project, the author managed to fit in an Ansible automation script that was done as a part of Automation of Services course by Jarmo Viinikanoja.

The script created virtual machines in Google's Cloud Platform as seen in Figure 28 and launched the authentication pipeline with HAProxy load balanced account registration website.



Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
connectionhandler1	europe-north1-a			10.166.15.236 (nic0)	35.228.234.20	SSH
loadbalancer1	europe-north1-a			10.166.15.194 (nic0)	35.228.9.7	SSH
loginserver1	europe-north1-a			10.166.15.196 (nic0)	35.228.89.240	SSH
mariadb1	europe-north1-a			10.166.0.63 (nic0)	35.228.216.187	SSH
restserver1	europe-north1-a			10.166.15.195 (nic0)	35.228.222.64	SSH
website1	europe-north1-a			10.166.15.192 (nic0)	35.228.197.0	SSH
website2	europe-north1-a			10.166.15.193 (nic0)	35.228.69.212	SSH

Figure 28. Ansible controller Google Cloud service

4 Results

The author succeeded in making a working containerized Login pipeline, allowing a user to register on a website and use those credentials when logging in to a game client prototype made with Unity. While making this pipeline, a protocol for the login service was written and implemented that can be used to create clients on different game-engines and programming languages.

Login pipeline is comprised of a very bland website hosted on top of HTTP and Python created by Django, which is not secure; however, it works for now. The webpage stores the credentials properly salted and hashed in to a MariaDB database. The login server component is created using Google's Go language, the author's self-made protocol and it is secured with the own security functions of Go language and HTTPS server using a proper TLS certification to encrypt communications.

A prototype of an online chat component was created using C/C++ and Epoll library that ran on a single thread on top of Linux Ubuntu operating system. The development of this component was halted due to time constraints concerning difficulties applying an encryption for it using LibreSSL or OpenSSL.

The author was able to create a networked Hardware rack fitted with 22 salvaged computers supposed to be the development platform for the whole system; however, due to time it was not possible to move the systems on top of it, even after using classes such as Cloud services and Automation of Services as a support for this part of the project.

This thesis was a large undertaking, and the author had to cut down the scope from the originally planned online game prototype and the system behind it in the end to make any sort of real progress and a report.

This thesis was a huge success in teaching the author about how to network different software securely using TCP and self-written protocols. The author also learned a great deal about different technologies required to host a system such as a multiplayer server on one’s own. The author feels very confident that in the future we can finish this project given enough time and further study.

5 Discussion

This project started with a very simple idea that quickly escalated out of hand. The very first idea was a small multiplayer environment that grew into a massively scalable multiplayer environment with a plethora of computers serving distinct roles inside the system.

In a way, this thesis was a perfect learning experience for the author, where one research lead to another research. The scale of the project also forced the author to go beyond just programming and try to and learn things on system level and design level. The author had to think about databases, protocols and hardware and even wood working that is really not taught in the software development orientation.

As the original goal was a working environment and a prototype, the game was simply not possible, which the author realized quickly after some time when designing the system UML diagram. This became more obvious as studies and part time work also had to be considered. Despite this overwhelming task, the author pressed on kept trying to the end.

References

10 reasons to migrate to MariaDB (if still using MySQL). 2015. Article in Seravo website. Accessed on 02 April 2020. Retrieved from https://seravo.fi/2015/10-reasons-to-migrate-to-mariadb-if-still-using-mysql

Adams, L. 2001. Ximian's Mono project: .NET for monkeys, penguins, and gnomes. Accessed on 04 April 2020. Retrieved from https://web.archive.org/web/20160410044339/http://www.techrepublic.com/article/ximians-mono-project-net-for-monkeys-penguins-and-gnomes/1044968/

Architecture. N.d. Article in Gluster Docs website. Accessed on 04 April 2020. Retrieved from https://docs.gluster.org/en/latest/Quick-Start-Guide/Architecture/

Avraham, B. 2017. What is REST — A Simple Explanation for Beginners, Part 1: Introduction. Accessed on 04 April 2020. Retrieved from https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-1-introduction-b4a072f8740f

Beginners Guide. 2019. Article in python website. Accessed on 02 April 2020. Retrieved from https://wiki.python.org/moin/BeginnersGuide/Overview

Ben-Kiki, O., Evans, C. & döt Net, I. 2009. YAML Ain’t Markup Language (YAML™) Version 1.2. Accessed on 01 April 2020. Retrieved from https://yaml.org/spec/1.2/spec.html

Chalin, P., Perkins, L. & Agung N. 2020. gRPC Concepts. Accessed on 02 April 2020. Retrieved from https://github.com/grpc/grpc.io/blob/master/content/docs/guides/concepts.md

Cox, M. 2018. Celebrating 20 Years of OpenSSL. Accessed on 01 April 2020. Retrieved from https://www.openssl.org/blog/blog/2018/12/20/20years/

Developer Guide. 2019. Article in Protocol Buffers website. Accessed on 02 April 2020. Retrieved from https://developers.google.com/protocol-buffers/docs/overview

Django at a glance. N.d. Article at django website. Accessed on 02 April 2020. Retrieved from https://docs.djangoproject.com/en/3.0/intro/overview/

Docker overview. N.d. Article in Docker documents website. Accessed on 01 April 2020. Retrieved from https://docs.docker.com/engine/docker-overview/

Docker vs Virtual Machine – Understanding the Differences. 2019. Article in Geekflare website. Accessed on 01 April 2020. Retrieved from https://geekflare.com/docker-vs-virtual-machine/

Ellingwood, J. 2015. How To Set Up Highly Available Web Servers with Keepalived and Floating IPs on Ubuntu 14.04. Accessed on 04 April 2020. Retrieved from https://www.digitalocean.com/community/tutorials/how-to-set-up-highly-available-web-servers-with-keepalived-and-floating-ips-on-ubuntu-14-04

Galera introduction. N.d. Video in Galera Cluster website. Accessed on 02 April 2020. Retrieved from https://galeracluster.com/videos/galera-introduction/

Games with Multiplayer tag released in 2018 in Steam platform. 2020. List in Pastebin website. Accessed on 01 April 2020. Retrieved from https://pastebin.com/pSTfChJK

Getting Started. N.d. Article in Visual Studio Code website. Accessed on 01 April 2020.Retrieved from https://code.visualstudio.com/docs

GlusterFS Documentation. N.d. Article in Gluster Docs website. Accessed on 04 April 2020. Retrieved from https://docs.gluster.org/en/latest/

Golang. 2017. Article in Computer Hope website. Accessed on 01 April 2020. Retrieved from https://www.computerhope.com/jargon/g/golang.htm

Haas, J. N.d. A History of the Unity Game Engine. Accessed on 04 April 2020. Retrieved from https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf

History of C++. N.d. Article in cplusplus website. Accessed on 01 April 2020. Retrieved from https://www.cplusplus.com/info/history/

How Ansible Works. N.d. Article in ansible website. Accessed on 01 April 2020. Retrieved from https://www.ansible.com/overview/how-ansible-works

Introduction to Vagrant. N.d. Article in Vagrantup website. Accessed on 01 April 2020. Retrieved from https://www.vagrantup.com/intro/index.html

ISO/IEC 14882:1998. 1998. Programming languages - C++. Status Withdrawn. Referenced 1.4.2020. Retrieved from https://www.iso.org/standard/25845.html

Lasn, I. 2019. Mermaid — Create Charts and Diagrams With Markdown-like Syntax. Accessed on 04 April 2020. Retrieved from https://medium.com/better-programming/mermaid-create-charts-and-diagrams-with-markdown-88a9e639ab14

LibreSSL. 2016. Article in Linux website. Accessed on 01 April 2020. Retrieved from https://www.linux.fi/wiki/LibreSSL

LibreSSL. 2019. Article in LibreSSL website. Accessed on 01 April 2020. Retrieved from https://www.libressl.org/

Linux Programmer's manual. 2019. Article about Epoll on man7 website. Accessed on 01 April 2020. Retrieved from http://man7.org/linux/man-pages/man7/epoll.7.html

Make a Videogame from Scratch - Tutorials. 2014. Playlist in Youtube website. Accessed on 04 April 2020. Retrieved from https://www.youtube.com/playlist?list=PL11F87EB39F84E292

MariaDB versus MySQL - Features. N.d. Article in MariaDB website. Accessed on 02 April 2020. Retrieved from https://mariadb.com/kb/en/mariadb-vs-mysql-features/

Most popular database technologies among developers worldwide as of January 2019. 2019. Statistic in Statista database. Accessed on 02 April 2020. Retrieved from https://www.statista.com/statistics/794187/united-states-developer-survey-most-wanted-used-database-technologies/

Most popular GitHub open source projects worldwide in 2019, by number of contributors (in 1,000s)*. 2019. Statistic in Statista database. Accessed on 02 April 2020. Retrieved from https://www.statista.com/statistics/824812/worldwide-github-popular-open-source-projects/

Number of active video gamers worldwide from 2014 to 2021. 2017. Statistic in Statista database. Accessed on 01 April 2020. Retrieved from https://www.statista.com/statistics/748044/number-video-gamers-world/

Number of games released on Steam worldwide from 2004 to 2018. 2019. Statistic in Statista database. Accessed on 01 April 2020. Retrieved from https://www.statista.com/statistics/552623/number-games-released-steam/

Pankoff, C. 2016. Reverse Engineering: Pokemon GO. Accessed on 02 April 2020. Retrieved from https://www.fknsrs.biz/blog/reverse-engineering-pokemon-go.html

Python Introduction. N.d. Article in w3schools website. Accessed on 02 April 2020. Retrieved from e

Results for dphttpd SMP. N.d. Article in epoll Scalability Web Page website. Accessed on 04 April 2020. Retrieved from http://lse.sourceforge.net/epoll/index.html

Roper, W. 2020. Python Remains Most Popular Programming Language. Accessed on 02 April 2020. Retrieved from https://www.statista.com/chart/21017/most-popular-programming-languages/

Schafer, C. 2019. Django Tutorials playlist in Youtube website. Accessed 20 April 2020. Retrieved from https://www.youtube.com/playlist?list=PL-osiE80TeTtoQCKZ03TU5fNfx2UY6U4p

Small. Simple. Secure. N.d. Article in alpineLinux website. Accessed on 01 April 2020. Retrieved from https://alpinelinux.org/about/

Srivastava, B. 2017. History Of C# Programming Language. Accessed on 04 April 2020. Retrieved from https://www.c-sharpcorner.com/blogs/history-of-c-sharp-programming-language

Team Avatar and the future of our prototype. 2012. Forum post in EVE General Discussion forums. Accessed on 04 April 2020. Retrieved from https://forums-archive.eveonline.com/default.aspx?g=posts&m=2023393

The story of Ubuntu. N.d. Article in ubuntu website. Accessed on 01 April 2020. Retrieved from https://ubuntu.com/about
e
Yadav, A. N.d. Go Programming Language (Introduction). Accessed on 01 April 2020. Retrieved from https://www.geeksforgeeks.org/go-programming-language-introduction/

Welcome to Ubuntu. N.d. Article in help.ubuntu.com website. Referenced in 1.4.2020. Retrieved from https://help.ubuntu.com/lts/installation-guide/s390x/ch01s01.html

What game engines do you currently use?. 2014. Statistic in Statista database. Accessed on 04 April 2020. Retrieved from https://www.statista.com/statistics/321059/game-engines-used-by-video-game-developers-uk/

What HAProxy is and isn't. N.d. Article in HAProxy Documentation website. Accessed on 04 April 2020. Retrieved from https://cbonte.github.io/haproxy-dconv/2.2/intro.html#3.1

What is Git. N.d. Article in Atlassian website. Accessed on 02 April 2020. Retrieved from https://www.atlassian.com/git/tutorials/what-is-git

Appendices

Appendix 1. Login protocol.

```
1. Login Handler Protocol

1.1. About

This protocol describes pre-agreed upon actions taken to assure
functionality
between a game client and a multiplayer game login service.

This protocol is a stateless binary protocol, as the whole proto-
col happens over
a single connections lifetime. Data is sent over- and read from
the network as
stream of bytes that are ordered as specified later in this docu-
ment.

1.2. Conventions and Terminology

A Single bit is 0 or 1 and a byte is 8 bits. Bytes are in Big en-
dian order, this
means that the most signifigant bit is first bit from left.

A Bit is represented as:

+---+
|  0|
+---+

A byte is represented as:

+---+---+---+---+---+---+---+---+
|128| 64| 32| 16|  8|  4|  2|  1|
+---+---+---+---+---+---+---+---+

Hexadecimal is 8 bits or 1 byte and are written as "0x0F". Hexa-
decimal values
are used to shorten longer byte representations as follows:

+----+
|0xFF|
+----+

Asterisk is used to mark side note somewhere below. *1 would be
explained little
later in the chapter, with a same mark, and number.

*1 Is a note later.

All strings are considered to be UTF-8 compatible.

All single bytes that form a concept, like name length (sinlge
byte in stream)
are considered unsigned 8-bit integers.
```

1.2.1. Terminology

connection: A data transfer line between two sockets.
client: Socket that starts the connection.
server: Socket that accepts and authenticates connections.
TLS: Transfer Layer Security.
Certificate: A key used to encrypt connections.
chunk: N amount of bytes that are connected by a concept.
GB: A Giga Byte. One bit to the power of 32. Simply: 2^32 bytes.
UTF-8: 8-bit Unicode Transformation Format.

1.3. Security

All connections should be made over TCP and secured by the server with modern
TLS encryption (as described in their RFC). All certificates should be secured
by a secure Root Certificate Authority.

All non-TLS connections should be rejected.

1.4. Messages

This protocol uses binary messages. Necessary chunk lengths are defined in the
messages themselves so no seperator characters or string parsing is needed.

Each message starts with a static chunk that identifies the message to this
protocol, which has these five following bytes:

```
+----+----+----+----+----+
|0x97|0x72|0xA8|0x9A|0x5B|
+----+----+----+----+----+
```

After these five bytes, there are two more unique bytes that identify the
type of the package.

1.4.1. Message types

Client can only send a single message to server, that is "Log in", to which the
log in server can respond with multiple different ways:

1. Ok!
2. Banned.
3. Error
    - 3.1. Generic error, no details.
    - 3.2. Database unreachable.
    - 3.3. REST server unreachable.
    - 3.4. No game time.
    - 3.5. Game not owned.

All these messages except 1. and 2. are error messages that stop the connection
attempt, but the application can still keep running.

1.4.2. Message declarations

1.4.2.1. Log in

Log in message is used by the multiplayer client to start the au-
thentication
process.

Log in message header looks as follows:

```
| Login Handler ID bytes |Log in ID| *1 |    *2   | *3 |    *4   |
+----+----+----+----+----+----+----+----+----+----+----+----+----+
|0x97|0x72|0xA8|0x9A|0x5B|0x61|0x7A|0x00|0x00|0x..|0x00|0x00|0x..|
+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

*1 Length of username bytes (unsigned 8 bit integer)
*2 Username bytes
*3 Length of password bytes (unsigned 8 bit integer)
*4 Password bytes

1.4.2.2. Ok

Server returns Ok message, when the login is successful. Ok mes-
sage has login
token and a connection handler port with it, so the client can
login to the game
server. The message looks like following:

```
| Login Handler ID bytes |  Ok ID  | *1 |    *2   | *3 |    *4   |
*5 | *6 |
+----+----+----+----+----+----+----+----+----+----+----+----+----
+----+----+
|0x97|0x72|0xA8|0x9A|0x5B|0x14|0x4A|0x00|0x00|0x..|0x00|0x00|0x..|
0x00|0x0.|
+----+----+----+----+----+----+----+----+----+----+----+----+----
+----+----+
```

*1 Length of token bytes (unsigned 8 bit integer)
*2 token bytes
*3 length of ip bytes (unsigned 8 bit integer)
*4 ip bytes
*5 Length of port bytes (unsigned 8 bit integer)
*6 port bytes

1.4.2.3. Banned

This message is a respond from the login server when the client
has been banned
from the service. It consists an ID and a short reason, that can
be displayed in
the game client.

```
| Login Handler ID bytes |   *1    |   *2    |    *3   |
+----+----+----+----+----+----+----+----+----+----+----+
|0x97|0x72|0xA8|0x9A|0x5B|0xA3|0xDB|0x25|0x25|0x00|0x..|
+----+----+----+----+----+----+----+----+----+----+----+
```

*1 Banned message ID
*2 Reason length (unsigned 16 bit integer)
*3 reason bytes

1.4.2.4. Error

Errors are returned by the login server in multiple different sit-
uations. Some
errors get called when internal systems fail and others when part
of the action
requested by the client fails.

```
| Login Handler ID bytes |   *1    |    *2   |
+----+----+----+----+----+----+----+----+----+
|0x97|0x72|0xA8|0x9A|0x5B|0x21|0x32|0xXX|0xXX|
+----+----+----+----+----+----+----+----+----+
```

```
*1 Internal error message ID
*2 Internal error ID, I.e:
    0x04, 0x3C for Generic error with no details.
    0x3C, 0xB3 for Database unreachable.
    0x9C, 0xEC for REST server unreachable.
    0x02, 0x25 for No game time.
    0xC7, 0xD0 for Game not owned.
```

(Internal error ID is 2 bytes)

Generic error is returned everytime something fails during login,
but more
information could compromise account security or system security.

In example Wrong username or bad password response could be used
by a hacker to
reorient their bruteforce attack. I.E. if you get bad password er-
ror, you know
then that the username exists and you can focus your bruteforce
just to try
rainbow attack on passwords. If you get bad username error, you
can just skip
rainbow attack part and try another username. Hiding these from
the login
process, makes the life of hacker a lot harder.

Getting a generic error response from server, will terminate the
log in attempt.